Université de Montréal

Programmes de génération et machines de Turing algébriques

par

Simon Pilette

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en informatique

Juin 2006

Université de Montréal

Faculté des études supérieures

Ce mémoire de maîtrise intitulé

Programmes de génération et machines de Turing algébriques

présenté par

Simon Pilette

a été évalué par un jury composé des personnes suivantes :

Président:	Gena Hahn
Directeur de recherche:	Pierre McKenzie
Membre:	Sylvie Hamel

Mémoire accepté le 6 septembre 2006

Sommaire

Ce mémoire est consacré à l'adaptation des programmes de génération pour les semi-anneaux et à l'étude de la machine de Turing algébrique.

Les programmes de génération sur un corps ont été définis par Karchmer et Wigderson dans [KW93]. Ils consistent à associer un espace vectoriel à chaque littéral sur un ensemble de variables. Une entrée est acceptée par un programme de génération si l'union des espaces vectoriels associés à ses littéraux contient un certain vecteur cible non nul. Bien qu'on ne puisse parler d'espace vectoriel, la définition et la sémantique d'un programme de génération peuvent être adaptées aux semi-anneaux. Nous étudierons, dans le second chapitre, la puissance expressive de tels programmes en fonction du semi-anneau sur lequel le programme est défini. Le problème d'évaluation d'un programme de génération sur un semi-anneau sera également étudié dans ce chapitre.

La machine de Turing algébrique a été introduite à des fins utilitaires par Damm *et al.* dans [DHM02]. Il semblait approprié d'amorcer une étude plus rigoureuse de ce modèle. Dans cette optique, le troisième chapitre de ce mémoire est consacré à l'adaptation de la théorie de la machine de Turing classique à sa variante algébrique. Ce chapitre contient également une définition de la machine de Turing algébrique alternante qui pourrait être un sujet intéressant pour une étude ultérieure.

Mots clés : *Programme de génération, semi-anneau, machine de Turing algébrique, complexité du calcul, programme de branchement algébrique.*

Abstract

This master thesis is devoted to the adaptation of the span programs for semirings and to the study of the algebraic Turing machine.

Span programs on a field were defined by Karchmer and Wigderson in [KW93]. They consist in associating a vector space to each literal. An input is accepted by a span program if the union of the vector spaces associated to its literals contains a certain non-zero target vector. Although one cannot speak about vector space for semirings, the definition and the semantics of a span program can be adapted to them. The second chapter consists in a study of the expressive power of such programs. An analysis for the evaluation problem is also available in this chapter.

The algebraic Turing machine was introduced by Damm *et al.* in [DHM02]. It seemed suitable to start a more rigorous study of this model. According to this, the third chapter of this master's thesis is dedicated to the adaptation of the theory of the traditional Turing machine to its algebraic version. This chapter also contains a definition of the alternating algebraic Turing machine which could be an interesting subject for a future study.

Key words : *Span Program, Semiring, Algebraic Turing Machine, Computational Complexity, Arithmetic Branching Program.*

Remerciements

Il y a définitivement beaucoup plus à apprendre pendant la maîtrise que ce que le mémoire peut contenir. Plusieurs personnes m'ont aidé à le comprendre.

Je remercie Marie, ma douce moitié, pour me donner le courage et l'inspiration de m'améliorer, toujours ... Mes parents et ma famille pour le soutient moral, les discussions nombreuses depuis toujours qui m'ont appris tant de choses. Je remercie Pierre, mon directeur de recherche, pour avoir su me superviser de manière habile et subtile. Merci également pour le soutien financier qui m'a permis de me soustraire de plusieurs soucis. Je remercie tous les amis à qui j'ai dit au moins une fois le mot maîtrise ces dernières années. En particulier, je pense à Bernard-Alexandre et à Mathieu qui sont toujours là pour moi.

À tous, bonne lecture.

Table des matières

Liste des figures

Liste des algorithmes

À ma famille

et à Marie ...

Chapitre 1

Introduction, définitions et motivations

1.1 Informatique théorique et complexité du calcul

Face à une tâche à accomplir, deux questions doivent être posées : est-il possible d'accomplir cette tâche ? si oui, puis-je y arriver avant le crépuscule de ma vie ? À la naissance de l'informatique, ces deux questions ont trouvé un sens plus théorique. Quelles sont les fonctions calculables par un ordinateur ? Quelles ressources seraient nécessaires à un ordinateur pour calculer une fonction ? Dans ce mémoire, nous nous concentrerons sur la seconde question.

Les théoriciens de l'informatique ont développé de nombreux outils pour analyser la complexité des langages $A \subseteq \Sigma^*$. Ici, Σ est un ensemble fini de symboles, appelé un alphabet, et Σ^* est l'ensemble des suites finies, appelées mots, de symboles de Σ. Un langage A sera dit décidable s'il existe une machine de Turing, i.e. un algorithme, permettant de déterminer si x est un élément de A et ce pour chaque mot $x \in \Sigma^*$. Pour un langage décidable, on cherchera à déterminer les ressources minimales, espace et temps, utilisées par une machine de Turing afin de le décider.

Bien qu'il y ait plusieurs alphabets Σ différents, on remarque qu'il est toujours possible d'encoder les éléments de Σ par des mots binaires de même longueur. Soit $\omega \in \Sigma^*$, notons $\omega_{BIN} \in \{0,1\}^*$ le mot obtenu en encodant, en binaire, les symboles de ω un à un. On peut donc associer, à un langage $A \subseteq \Sigma^*$, sa version binaire, A_{BIN}, qui

contient les mots ω_{BIN} tels que $\omega \in A$. Ainsi, nous limiterons notre étude aux langages sur l'alphabet binaire.

1.2 Petit lexique de la complexité du calcul

La présente section permettra au lecteur de se familiariser avec les notions, notations et conventions utilisées dans ce mémoire.

La fonction caractéristique d'un langage A sur l'alphabet $\{0,1\}$ est une fonction binaire, $\chi_A : \{0,1\}^* \to \{0,1\}$, qui prend la valeur 1 exactement pour les mots x appartenant à A. Ainsi, décider un langage A revient à calculer la fonction χ_A.

Définition 1. *Soit $n \in \mathbb{N}$, $A \subseteq \{0,1\}^*$ et $x \in \{0,1\}^n$, alors :*

1. *x est dit de longueur n,*
2. *ε est le mot vide, c'est-à-dire le seul mot de longueur 0 sur l'alphabet $\{0,1\}$,*
3. *$\chi_A^n : \{0,1\}^n \to \{0,1\}$ est la restriction de χ_A aux mots de longueur n.*

On remarque que la fonction caractéristique d'un langage sur l'alphabet binaire est une fonction binaire et que la préimage de 1 d'une fonction binaire est un langage binaire. Nous ne ferons donc pas de distinction entre décider un langage sur l'alphabet binaire et calculer une fonction binaire.

Soit $n, k \in \mathbb{N}$, $x \in \{0,1\}^n$ et $f : \{0,1\}^n \to \{0,1\}^k$, on peut voir x comme un mot de longueur n sur l'alphabet $\{0,1\}$ ou comme un argument pour la fonction f. Si x est considéré comme un mot alors on note $x[i]$ le i-ème symbole de x.

Définition 2. *Soit $n, k \in \mathbb{N}$ et $f : \{0,1\}^n \to \{0,1\}^k$, alors :*

1. *la fonction f est dite d'arité n,*
2. *si $k = 1$, la fonction $\overline{f} = 1 - f$ est appelée complément de f,*
3. *$\{x_1, x_2, \ldots, x_n\}$ sont les n variables de f,*
4. *$X_n := \{x_1, x_2, ...x_n, \overline{x_1}, \overline{x_2}, ..., \overline{x_n}\}$ est l'ensemble des littéraux sur n variables,*
5. *$[n] := \{1, 2, 3, \ldots, n\}$.*

Nous utiliserons parfois la notation $x \in \{0,1\}^n$, il faut alors comprendre que $n \in \mathbb{N}$.

Définition 3. *Soit $n \in \mathbb{N}$, $i \in [n]$, x_i une variable binaire, $\sigma \in \{0,1\}$ et $y \in \{0,1\}^n$, alors :*

1. $x_i^\sigma = \overline{x_i}$ *si* $\sigma = 0$ *et* $x_i^\sigma = x_i$ *si* $\sigma = 1$,
2. y *satisfait* x_i^σ *si et seulement si* $y[i] = \sigma$.

Soit $G = (V, E)$ un graphe orienté, alors l'entrance (resp. sortance) d'un sommet $v \in V$ est le nombre d'arcs entrants dans (resp. sortants de) v.

1.3 Structures algébriques

Ce mémoire est consacré à plusieurs modèles algébriques de calcul. Par mesure de complétude, cette section présente de manière succincte les différentes structures algébriques qui seront utilisées par les modèles de calcul. Le manuel d'algèbre linéaire de Leroux [Ler83], le livre de Kuich et Salomaa [KS86] ainsi que les notes de cours de Pierre McKenzie [McK04] ont servi de référence.

Soit E, un ensemble contenant 0, et $+, \times : E \times E \to E$, deux applications internes, alors on peut définir les cinq propriétés suivantes :

1. $(\forall_{a,b,c \in E})\ a + (b + c) = (a + b) + c$ (Associativité),
2. $(\forall_{a \in E})\ 0 + a = a = a + 0$ (Neutre),
3. $(\forall_{a \in E})\ (\exists_{(-a) \in E})\ a + (-a) = 0 = (-a) + a$ (Inverse),
4. $(\forall_{a,b \in E})\ a + b = b + a$ (Commutativité),
5. $(\forall_{a,b,c \in E})\ a \times (b + c) = (a \times b) + (a \times c)$ (Distributivité).

Ces propriétés permettent de définir les différentes structures algébriques que nous utiliserons dans ce mémoire. On notera E^0 l'ensemble $E \setminus \{0\}$.

Définition 4.

1. $(E, +)$ *est un monoïde si 1 et 2 sont vérifiées*
2. $(E, +)$ *est un groupe si 1, 2 et 3 sont vérifiées*
3. $(E, +)$ *est commutatif si 4 est vérifiée*
4. $(E, +, \times)$ *est un semi-anneau si :*

- $(E, +)$ *est un monoïde commutatif*
- $(E^0, \times)$ *est un monoïde avec 1 comme neutre*
- $0 \times a = a \times 0 = 0$
- *5 est vérifiée*

5. $(E, +, \times)$ *est un corps si :*
   - $(E, +)$ *est un groupe commutatif*
   - $(E^0, \times)$ *est un groupe avec 1 comme neutre*
   - $0 \times a = a \times 0 = 0$
   - *5 est vérifiée*

Voici quelques exemples de ces structures :

- $(\mathbb{N}, +)$ est un monoïde commutatif,
- $(\mathbb{R}^0, \times)$ est un groupe commutatif,
- $(\mathbb{N}, +, \times)$ est un semi-anneau,
- un corps $\mathbb{K}$ est un semi-anneau,
- $(\mathbb{R}, +, \times)$ est un corps.

1.4 Définition des modèles de calculs

Dans cette section, nous introduisons les modèles algébriques de calcul étudiés dans ce mémoire.

1.4.1 Les programmes de génération

Le premier modèle de calcul étudié dans ce mémoire est le programme de génération . Il s'agit d'un modèle algébrique de calcul qui a été introduit par Karchmer et Wigderson dans [KW93]. Un programme de génération permet de calculer une fonction binaire d'arité $n \in \mathbb{N}$. Dans leur article, Karchmer et Wigderson ont limité leur recherche aux programmes de génération sur un corps. Leur principal résultat est la démonstration que les programmes de génération sur $GF(2)$ peuvent simuler les programmes de branchement avec parité (section 1.4.2) comme mode d'acceptance en doublant la taille. Ce résultat a été raffiné par Beimel et Gál dans [BG99].

Ces derniers ont défini les programmes de branchement algébrique sur un corps et démontré qu'un programme de branchement algébrique sur un corps peut toujours être simulé par un programme de génération, avec une taille raisonnable, sur le même corps. Il est à noter que Beimel, dans sa thèse de doctorat [Bei96], a également développé un argument simple démontrant l'équivalence entre les programmes de génération monotones et les schémas linéaires de partage de secret, objet d'étude du domaine de la cryptographie que nous n'aborderons pas dans ce mémoire. Notons que Karchmer et Widgerson avaient déjà donné une construction pour obtenir un schéma linéaire de partage de secret à partir d'un programme de génération monotone.

Voici la définition formelle d'un programme de génération.

Définition 5. *Un programme de génération (« span program ») sur un corps* $\mathbb{K}$ *(*$\mathbb{K}$*-SP) est un triplet* $\hat{M} = (M, \mu, \tau)$ *où :*

- $M \in \mathbb{K}^{k \times m}$ *est une matrice avec* k *lignes,* m *colonnes et dont les entrées sont prises dans* $\mathbb{K}$*,*
- $\mu : \{i | i \in [m]\} \rightarrow X_n$ *est une fonction qui associe un littéral à chaque colonne* M_i *de* M*,*
- $\tau \in \mathbb{K}^{1 \times k}$ *est un vecteur non-nul appelé le vecteur cible.*

Pour une entrée $x \in \{0,1\}^n$, on note M^x la sous-matrice de M qui contient les colonnes de M consistantes avec x. C'est-à-dire qu'une colonne M_i de M est une colonne de M^x lorsque $\mu(i)$ est satisfait par x. On note que μ peut également être vu comme un vecteur sur X_n. La composante i de μ est simplement $\mu(i)$. Considérer μ comme un vecteur sera très utile pour encoder ce dernier.

Nous pouvons maintenant définir le critère d'acceptation d'un programme de génération $\hat{M}$ pour une entrée x. On dira que l'entrée $x \in \{0,1\}^n$ est acceptée par $\hat{M}$ si et seulement si τ est une combinaison linéaire, sur $\mathbb{K}$, des colonnes de M^x ; on notera ceci $\tau \in \langle M^x \rangle$. On définit $f_{\hat{M}} : \{0,1\}^n \rightarrow \{0,1\}$, la fonction binaire calculée par le programme de génération $\hat{M}$, par :

$$f_{\hat{M}}(x) = 0 \Leftrightarrow \tau \notin \langle M^x \rangle.$$

Nous avons transposé la matrice M, utilisée dans [KW93], afin de souligner qu'évaluer un programme de génération sur une entrée $x \in \{0,1\}^n$ est en fait équivalent à résoudre

un système d'équations linéaires. En effet, il est clair que $\tau \in \langle M^x \rangle$ si et seulement si il existe un vecteur $y \in \mathbb{K}^{k_x}$, où k_x est le nombre de colonnes de M^x, et $M^x \cdot y^T = \tau^T$.

Définition 6. *Soit $\hat{M} = (M, \mu, \tau)$, un programme de génération avec k lignes, m colonnes et soit E, la plus grande entrée de M, alors la taille de $\hat{M}$ est $k \cdot m \cdot lg\ E$.*

Cette définition diffère de la définition de taille utilisée dans [KW93]. Nous verrons dans le chapitre 2 de ce mémoire que chaque élément de cette définition est nécessaire pour les programmes de génération sur un semi-anneau.

Soit $f : \{0,1\}^n \to \{0,1\}$, une fonction binaire d'arité $n \in \mathbb{N}$, alors $\mathtt{SP}_{\mathbb{K}}(f)$ est la taille minimale d'un programme de génération sur $\mathbb{K}$ calculant f.

On remarque que la mesure proposée est polynomialement reliée à la mesure de taille de [KW93], soit le nombre de colonnes, pour les programmes de génération minimaux sur un corps.

1.4.2 Les programmes de branchement

Afin de bien comprendre les résultats de Beimel et Gál ([BG99]), il est nécessaire d'introduire les programmes de branchement ainsi que leur extension algébrique.

Lee fut le premier à étudier les programmes de branchement. Dans son article [Lee59], il s'inspire de la formule de décomposition de Shannon :

$$f \equiv x_i \wedge f_{x_i=1} + \overline{x_i} \wedge f_{x_i=0},$$

afin d'utiliser les programmes de branchement comme une représentation, ou structure de données, pour les fonctions binaires.

C'est le travail de Akers [Ake78] qui permet aux programmes de branchement, aussi appelés diagrammes de décision binaire, d'atteindre une renommée auprès des ingénieurs. En particulier, les diagrammes de décision binaire sont utilisés en vérification de matériel («model checking»).

Un programme de branchement est un graphe orienté acyclique, G, possédant un unique sommet d'entrance nulle, la source, et un unique sommet de sortance nulle, la cible. Tous les sommets de G, sauf la cible, sont étiquetés par un littéral de X_n et ont sortance 1 ou 2. Les arcs sortants d'un sommet sont étiquetés par 0 ou 1 et un sommet ne peut posséder deux arcs sortants étiquetés par la même constante. La fonction

binaire calculée par un programme de branchement prend la valeur 1 exactement pour les entrées $x \in \{0,1\}^n$ telles que G^x, le sous-graphe de G ne contenant que les arcs consistants avec x, possède un chemin de la source à la cible. La taille d'un programme de branchement est le nombre de sommets de G.

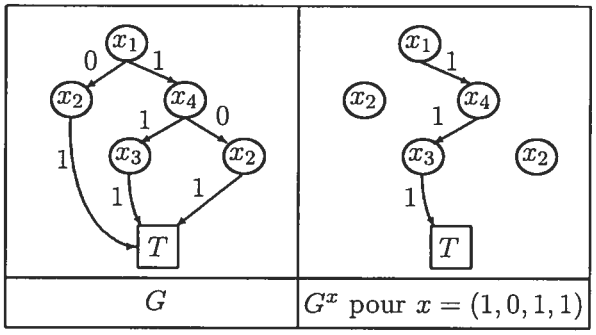

FIG. 1.1 – Un programme de branchement tel que $f(1,0,1,1) = 1$

Les programmes de branchement ont largement été étudiés depuis 1980. En particulier, il existe plusieurs restrictions de ce modèle pour lesquelles des bornes inférieures exponentielles sur la taille sont connues. Soulignons le résultat de Barrington [Bar89] qui a démontré que les programmes de branchement de largeur constante et de taille polynomiale capturent exactement la classe NC^1. Razborov a rassemblé quelques résultats dans son article [Raz91]. Pour une revue plus approfondie, l'excellente monographie de Wegener [Weg00] couvre en grande partie les aspects théoriques et pratiques liés aux programmes de branchement.

Les programmes de branchement ont intéressé les théoriciens de l'informatique à la suite des travaux de Cobham [Cob66] qui a démontré que la taille d'un programme de branchement, calculant une fonction binaire f, est liée à l'espace utilisé par une machine de Turing qui calcule la même fonction binaire. La section 1.6 rappelle ce lien bien connu.

En 1999, suivant les travaux de Karchmer *et al.*, Beimel et Gàl ont généralisé les programmes de branchement dans le but d'étendre les résultats de [KW93].

Définition 7 ([BG99]). *Un programme de branchement algébrique (« arithmetic branching program ») sur un corps $\mathbb{K}$ ($\mathbb{K}$-AP) est un quintuplet $B = (G, \mu, \omega, s, t)$ où :*

- *$G = (V, E)$ est un graphe orienté et acyclique,*
- *$\mu : E \to X_n$ est une fonction qui associe un littéral à chaque arête,*

- $\omega : E \to \mathbb{K}$ *est une fonction de poids pour les arêtes,*
- $s \in S$ *est la source du programme de branchement algébrique,*
- $t \in S$ *est la cible du programme de branchement algébrique.*

Pour une entrée $x \in \{0,1\}^n$, on note $G^x = (V, E^x)$ le sous-graphe de G contenant tous les sommets de G ainsi que les arêtes $e \in E$ telles que x satisfait $\mu(e)$. On note Φ_x l'ensemble des chemins ϕ de s à t dans G^x. À chaque chemin $\phi \in \Phi_x$, on associe un poids, $W(\phi)$, qui est le produit des $\omega(e_j)$ tels que e_j est un arc du chemin ϕ. On définit la fonction $f_B^{\mathbb{K}} : \{0,1\}^n \to \mathbb{K}$ qui associe à x un poids dont la valeur est :

$$f_B^{\mathbb{K}}(x) = \sum_{\phi \in \Phi_x} W(\phi).$$

Nous pouvons maintenant définir le critère d'acceptation d'un programme de branchement algébrique B pour une entrée x. On dira que $x \in \{0,1\}^n$ est acceptée par B si et seulement si $f_B^{\mathbb{K}}(x)$ est non-nul. On définit $f_B : \{0,1\}^n \to \{0,1\}$, la fonction binaire calculée par le programme de branchement algébrique B, par :

$$f_B(x) = 0 \Leftrightarrow f_B^{\mathbb{K}}(x) = 0.$$

La taille d'un programme de branchement algébrique $B = (G, \mu, \omega, s, t)$ est le nombre de sommets du graphe G.

Soit $f : \{0,1\}^n \to \{0,1\}$, une fonction binaire d'arité n, alors $\mathtt{ABP}_{\mathbb{K}}(f)$ est la taille minimale d'un programme de branchement algébrique sur $\mathbb{K}$ calculant f.

Il est maintenant possible d'énoncer formellement la proposition de Karchmer et Wigderson.

Définition 8. *Un programme de branchement avec mode d'acceptation parité est un programme de branchement algébrique sur $GF(2)$.*

Proposition 9 ([KW93]). *Soit $f : \{0,1\}^n \to \{0,1\}$, alors* $\mathtt{SP}_{\mathtt{GF(2)}}(f) \leq 2 \cdot \mathtt{ABP}_{\mathtt{GF(2)}}(f)$.

La section suivante traite de la généralisation de cette proposition que Beimel et Gál ont démontrée. Ils ont en particulier démontré que la proposition précédente était réciproque pour tous les corps finis. C'est-à-dire que pour tout corps $\mathbb{K}$, la taille minimale d'un programme de génération, sur $\mathbb{K}$, calculant une fonction binaire f est polynomialement reliée à la taille minimale d'un programme de branchement algébrique, sur $\mathbb{K}$, calculant f.

1.4.3 Relations entre les programmes de branchement algébrique et les programmes de génération

Dans sa thèse de doctorat, Anna Gál [Gal95] a démontré que si une fonction monotone f est calculée par un programme de génération, de taille S, sur un corps, alors $\overline{f}$ est calculée par un programme de génération de taille S sur le même corps. Ce résultat a ensuite été étendu à toutes les fonctions binaires par Pudlák et Sgall [PS98]. En effet, en utilisant la dualité de la programmation linéaire, ces derniers ont démontré que si une fonction f est calculée par un programme de génération de taille S alors $\overline{f}$ est calculée par un programme de génération de taille $O(n \cdot S)$, où n est l'arité de f.

Beimel et Gál, dans [BG99], ont ensuite développé un argument élégant visant à démontrer que pour tout corps $\mathbb{K}$ et pour tout programme de branchement algébrique B de taille S sur $\mathbb{K}$, il existe un programme de génération $\hat{M}$, sur $\mathbb{K}$, tel que $f_B \equiv \overline{f_{\hat{M}}}$ et tel que la taille de $\hat{M}$ est $2 \cdot S$. Ils ont donc conclu, par [PS98], qu'il existe un programme de génération $\hat{M}'$, de taille $O(2 \cdot n \cdot S)$ calculant f_B.

Il est naturel de se demander si la simulation inverse est également possible. La question est posée et partiellement solutionnée dans leur article.

Proposition 10 ([BG99]). *Pour tout corps $\mathbb{K}$ et fonction $f : \{0,1\}^n \to \{0,1\}$, les énoncés suivants sont équivalents :*

1. *Il existe une constante c telle que $\mathtt{ABP}_{\mathbb{K}}(f) \in O(\mathtt{SP}_{\mathbb{K}}(f)^c)$,*
2. *Il existe une constante c telle que $\mathtt{ABP}_{\mathbb{K}}(\overline{f}) \in O(\mathtt{ABP}_{\mathbb{K}}(f)^c)$,*
3. *Il existe une constante c telle que pour toute matrice $A(x)$ sur $\mathbb{K}$ avec k colonnes et tout $r \in \mathbb{N}$, il existe un programme de branchement algébrique de taille $O(k^c)$ sur $\mathbb{K}$ qui accepte $x \in \{0,1\}^n$ si et seulement si le rang de $A(x)$ est r.*

Une matrice $A(x)$ est simplement une matrice avec des entrées qui sont des polynomes d'une variable, x, sur $\mathbb{K}$. Dans la preuve qui suit, celle l'équivalence entre les deux premiers énoncés de 10 sera utilisée. Le troisième énnoncé est donné simplement par souci de complétude.

Corollaire 11 ([BG99]). *Si $\mathbb{K}$ est un corps fini alors pour toute fonction binaire f, il existe une constante $c \in \mathbb{N}$ telle que $\mathtt{ABP}_{\mathbb{K}}(f) \in O(\mathtt{SP}_{\mathbb{K}}(f)^c)$.*

Démonstration. Par définition, connecter la cible d'un programme de branchement algébrique A à la source d'un programme de branchement algébrique B par un arc de poids 1 donne un nouveau programme de branchement calculant la fonction $f_A^{\mathbb{K}} \cdot f_B^{\mathbb{K}}$.

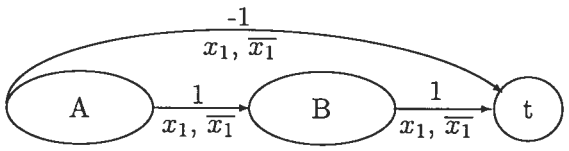

FIG. 1.2 – Un programme calculant $(-1) + f_A^{\mathbb{K}} \cdot f_B^{\mathbb{K}}$

Un groupe G est dit cyclique s'il existe un élément $\alpha \in \mathbb{K}$, appelé générateur du groupe, tel que

$$G = \{1 = \alpha^{q-1}, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{q-2}\},$$

où $|G| = q-1$. On sait que le groupe multiplicatif d'un corps fini $\mathbb{K} = GF(q)$ est cyclique et que l'ordre de ce groupe est $q-1$. On obtient dès lors que

$$x^{q-1} = (\alpha^{i_x})^{q-1} = (\alpha^{q-1})^{i_x} = 1^{i_x} = 1$$

pour tout $x \in \mathbb{K}^*$.

Supposons maintenant que A est un programme de branchement sur $\mathbb{K}$ qui calcule f, notons A_q le programme de branchement consistant en $q-1$ copies de A connectées en série par des arcs de poids 1. On observe que $f_{A_q}^{\mathbb{K}}(x) = (f^{\mathbb{K}}(x))^{q-1} = f(x)$. Soit B un programme de branchement algébrique sur $\mathbb{K}$ consistant en A_q auquel on ajoute un arc de s à t de poids -1, alors $f_B^{\mathbb{K}}(x) = -1 + f(x)$ et donc $f_B \equiv \overline{f}$. Puisque $|B| = (q-1) \cdot |A| + 1$, on conclut que $\mathtt{ABP}_{\mathbb{K}}(\overline{f}) \in O((q-1) \cdot \mathtt{ABP}_{\mathbb{K}}(f))$ et on termine en appliquant la proposition 10. ■

Il est à noter que si $\mathbb{Q}$ vérifie un des critères de la proposition 10 alors la classe $C_=L$ est fermée sous complémentation ([BG99]). Rappelons que la classe $C_=L$ est la classe des langages L tels qu'il existe une machine non-déterministe de Turing M utilisant un espace logarithmique et telle que pour une entrée $x \in \{0,1\}*$, le nombre de chemins acceptants de M est égal au nombre de chemins rejetants de M si et seulement si $x \in L$.

1.4.4 La machine de Turing algébrique

La machine de Turing est certainement le modèle théorique de calcul le plus célèbre. Introduite par Alan Turing en 1936 [Tur36], elle a permis de clarifier la notion intuitive d'algorithme. Plusieurs variantes de la machine de Turing ont été définies depuis. Le chapitre 3 sera consacré à l'étude de l'une des variantes présentes dans la littérature récente : la machine de Turing algébrique.

Introduite par Damm *et al.* dans [DHM02] , la machine de Turing algébrique est une généralisation naturelle des programmes de branchement algébrique de Beimel et Gál. En effet, il s'agit d'une machine de Turing non-déterministe munie d'une fonction de poids associant une valeur prise dans un semi-anneau à chaque transition. Plus formellement, la machine de Turing algébrique est définie comme suit :

Définition 12. *Une machine de Turing algébrique sur un semi-anneau S (S-MTA) est un septuplet $M = (Q, \Sigma, \Gamma, \delta, q_0, B, q_f)$ où :*

- *Q est un ensemble d'états,*
- *Σ est l'alphabet d'entrées,*
- *Γ est l'alphabet du ruban, $\Sigma \subseteq \Gamma$,*
- *$q_0 \in Q$ est l'état initial,*
- *$B \in \Gamma$ est le symbole blanc (case vide),*
- *$q_f \in Q$ est l'état final,*
- *δ est une relation de transition de la forme :*

$$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, H, R\} \times S.$$

Essentiellement, la machine de Turing algébrique, sur S, est une machine non-déterministe qui associe un poids $s \in S$ à chaque transition.

Définition 13. *Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, B, q_f)$ une machine de Turing algébrique, alors une configuration de M est un triplet $(u, q_i, v) \in \Gamma^* \times Q \times \Gamma^*$. Une telle configuration signifie que M contient uv sur son ruban, se trouve dans l'état q_i et que la tête de M est au-dessus du premier symbole de v.*

La configuration d'une machine de Turing algébrique, sur une branche de calcul, est suffisante pour déterminer les prochaines étapes, non-déterministes, du calcul. Une

étape de calcul consiste en le passage d'une configuration C_i à une configuration C_j suivant la relation de transition δ.

Définition 14. *Soit* $M = (Q, \Sigma, \Gamma, \delta, q_0, B, q_f)$ *une* S-MTA, $a, b, c, d \in \Gamma$, $C_1 = (ua, q_1, bcv)$, C_2 *deux configurations telles que* $u, v \in \Gamma^*$ *et* $s \in S$. *Il existe une transition de* C_1 *à* C_2 *si et seulement si :*

1. $(q_1, b, q_2, d, D, s) \in \delta$ *et* $C_2 = (uad, q_2, cv)$

   ou

2. $(q_1, b, q_2, d, G, s) \in \delta$ *et* $C_2 = (u, q_2, adcv)$

   ou

3. $(q_1, b, q_2, d, H, s) \in \delta$ *et* $C_2 = (ua, q_2, dcv)$.

S'il existe une transition de la configuration C_1 à la configuration C_2, on dit que C_1 mène à C_2. Une transition de C_1 à C_2 est notée $C_1 \vdash C_2$. Une configuration, C, est dite accessible de C' s'il existe une suite finie de configurations, $(C_i)_{i\in[k]}$, telle que :

$$C' \vdash C_1 \vdash C_2 \vdash \cdots \vdash C_k \vdash C.$$

La configuration initiale de M sur une entrée $x \in \{0,1\}^*$ sera notée $C_0(x)$. On ne considère que les machines de Turing algébriques effaçant leur ruban d'entrée avant d'entrer dans l'état final. Ainsi on peut parler d'une unique configuration finale : $C_f(x) = (\epsilon, q_f, \epsilon)$. De plus, on ne considère que les machines de Turing algébriques qui s'arrêtent sur toutes les branches de leur calcul.

Pour une entrée $x \in \{0,1\}^n$, on note $G^x = (V^x, E^x)$ le graphe orienté et pondéré des configurations visitées durant le calcul de M sur x. Plus formellement, V^x est donc l'ensemble des configurations de M accessibles à partir de $C_0(x)$ et E^x est l'ensemble des arcs (C_1, C_2) tel que C_1 mène à C_2. On définit également une fonction de poids, $\omega : \{(C_1, C_2) \mid (C_1, C_2) \in E^x\} \to S$, qui associe le poids de la transition $C_1 \vdash C_2$ à l'arc (C_1, C_2). On note que ce graphe n'est pas nécessairement calculable efficacement mais qu'il existe et est bien défini.

On note Φ_x l'ensemble des chemins ϕ de $C_0(x)$ à $C_f(x)$ dans G^x. À chaque chemin $\phi \in \Phi_x$, on associe un poids, $W(\phi)$, qui est le produit des $\omega(e_j)$ tels que e_j est un arc du chemin ϕ. Il est important d'effectuer le produit selon l'ordre induit par le chemin

car la commutativité de la multiplication n'est pas supposée. On définit la fonction $f_M^S : \{0,1\}^* \to S$ qui associe, à x, un poids dont la valeur est :

$$f_M^S(x) = \sum_{\phi \in \Phi_x} W(\phi).$$

Nous pouvons maintenant définir le critère d'acceptation d'une machine de Turing algébrique M pour une entrée x. On dira que $x \in \{0,1\}^*$ est acceptée par M si et seulement si $f_M^S(x)$ est non-nulle. On définit $f_M : \{0,1\}^* \to \{0,1\}$, la fonction binaire calculée par la machine de Turing algébrique M par :

$$f_M(x) = 0 \Leftrightarrow f_M^S(x) = 0.$$

Deux mesures de complexité sont généralement considérées pour la machine de Turing : le temps de calcul et l'espace de calcul. Dans les deux cas, on ne considère que les machines de Turing algébriques dont tous les chemins sont de longueur finie. En particulier, G^x est toujours supposé acyclique.

Pour une entrée $x \in \{0,1\}^*$, le temps de calcul de M sur x, noté $time_M(x)$, est la longueur du plus long chemin de G^x.

Définition 15. *Le temps de calcul d'une machine de Turing algébrique M est une fonction $t : \mathbb{N} \to \mathbb{N}$ telle que $t(n) = \max\{\ time_M(x) \mid x \in \{0,1\}^n\ \}$.*

Pour définir l'espace de calcul d'une machine de Turing algébrique M fonctionnant en espace moindre que linéaire, il est nécessaire de la munir de deux rubans. Le premier, en lecture seule, est le ruban d'entrée ; le second, en lecture et écriture, est le ruban de travail. Une configuration pour une telle machine devient donc un tuple $(p, u, q_i, v) \in [n] \times \Gamma^* \times Q \times \Gamma^*$, le p désignant la position de la tête de lecture. La configuration initiale devient donc $C_0(x) = (1, \epsilon, q_0, \epsilon)$ et la configuration finale devient $C_f = (1, \epsilon, q_f, \epsilon)$. On remarque en particulier que $C_0(x)$ ne dépend plus de x. De plus, on remarque que le contenu du ruban d'entrée n'est pas représenté dans une configuration. Nous utiliserons la notation $C_1 \vdash^x C_2$ pour signifier qu'il existe une transition de la configuration C_1 à la configuration C_2 qui est consistante avec $x \in \{0,1\}^*$. Dans le graphe G^x, on note le poids d'un arc (C_1, C_2) par $\omega(C_1, C_2, x)$ afin de bien signifier que le x n'est pas implicite.

Définition 16. *L'espace de calcul d'une machine du Turing algébrique M avec un ruban dédié à l'entrée et un ruban dédié au calcul est une fonction $s : \mathbb{N} \to \mathbb{N}$ telle que $s(n)$ est*

le nombre maximal de cellules utilisées sur le ruban lors du calcul de M sur une entrée $x \in \{0,1\}^n$.

Voici la définition de plusieurs classes de langages et fonctions que nous utiliserons dans le présent mémoire.

Définition 17. *Pour un semi-anneau S :*

- $S\text{-}TIME(t(n)) := \{f_M \mid M$ *est une* S-MTA *fonctionnant en temps* $O(t(n))\}$,
- $S\text{-}SPACE(s(n)) := \{f_M \mid M$ *est une* S-MTA *fonctionnant en espace* $O(s(n))\}$,
- $S\text{-}\#TIME(t(n)) := \{f_M^S \mid M$ *est une* S-MTA *fonctionnant en temps*$O(t(n))\}$,
- $S\text{-}\#SPACE(s(n)) := \{f_M^S \mid M$ *est une* S-MTA *fonctionnant en espace*$O(s(n))\}$,
- S-L $= S\text{-}SPACE(log(n))$,
- S-P $= \bigcup_{k\in\mathbb{N}} S\text{-}TIME(n^k)$,
- S-#P $= \bigcup_{k\in\mathbb{N}} S\text{-}\#TIME(n^k)$.

Les classes S-P et S-L sont l'analogue des classes NP et NL définies par la machine de Turing non-déterministe. Nous vous référons au manuel de Sipser [Sip05] pour un rappel de ces classes. Mentionnons aussi que la page internet située à l'adresse

« `http://qwiki.caltech.edu/wiki/Complexity%5FZoo` »

contient une brève description de la presque totalité des classes étudiées par les informaticiens.

La machine de Turing algébrique permet de caractériser plusieurs classes de langages et de fonctions de manière élégante. La proposition suivante, énoncée sans preuve dans l'article de Damm *et al.*, permet de constater qu'il suffit parfois de changer de semi-anneau afin de capturer une classe différente.

Proposition 18 ([DHM02]).

1. $\mathbb{B}$-P $= \mathbb{N}$-P $=$ NP,
2. $\mathbb{Z}_q$-P $= MOD_q$-P,
3. $GF(2)$-P $= \oplus$P,
4. $\mathbb{N}$-#P $=$ #P,
5. $\mathbb{Z}$-#P $=$ *Gap*P.

La classe de fonctions #P a été introduite par Valiant dans [Val79]. Une fonction $f : \Sigma \to \mathbb{N}$ est élément de #P s'il existe une machine de Turing non-déterministe M telle que pour toute entrée $x \in \Sigma$ le nombre de chemins acceptant de M est exactement $f(x)$. Le problème complet classique pour #P est #SAT qui compte le nombre d'affectations satisfaisant une formule booléenne donnée comme argument. Une définition de la notion de complétude est donnée à la section 2.3.2. Fenner *et al.* [FFK91] ont introduit la classe GAP qui est la fermeture de #P sous la soustraction, c'est-à-dire que GAP contient exactement toutes les fonctions pouvant s'exprimer comme la différence de deux fonctions de #P. On peut également voir la classe GAP comme la classe des fonctions f telles qu'il existe une machine de Turing non-déterministe dont la différence entre le nombre de chemins acceptants et de chemins rejetants sur une entrée x est $f(x)$.

Voici la démonstration de (5.). Les autres démonstrations sont similaires et moins intéressantes. On utilisera l'énoncé (4.) de la proposition dans la démontration.

Démonstration. ($\supseteq$) Soit $h : \mathbb{N} \to \mathbb{N} \in$ GAP, alors $h \equiv f - g$ où f et g sont des fonctions de #P ($=$ $\mathbb{N}$-#P). Soit M_f (resp. M_g) une machine de Turing algébrique sur $\mathbb{N}$ fonctionnant en temps polynomial et calculant f (resp. g). Considérons M_h, la $\mathbb{Z}$-MTA donnée par l'algorithme suivant :

Algorithme 1 Description de M_h.

1: M_h : sur entrée $x \in \{0,1\}^*$
2: Sans bouger la tête : se connecter sur l'état initial de M_f avec poids 1 et sur l'état initial de M_g avec poids -1.
3: Simuler M_f et M_g sur x avec les mêmes poids de transition.

On a alors $f^{\mathbb{Z}}_{M_h}(x) = f^{\mathbb{N}}_{M_f}(x) - f^{\mathbb{N}}_{M_g}(x) = f(x) - g(x) = h(x)$. Puisque M_f et M_g fonctionnent en temps polynomial, la machine M_h fonctionne également en temps polynomial. On obtient donc que $h \in \mathbb{Z}$-#P.

($\subseteq$) Soit $h : \mathbb{N} \to \mathbb{N} \in \mathbb{Z}$-#P et M_h une $\mathbb{Z}$-MTA calculant h en temps polynomial. Considérons M_f (resp. M_g), qui agit comme M_h mais dont le poids d'une transition est $|s|$ si la transition correspondante, dans M_h, a un poids s. Le signe du produit des poids,

pour un chemin particulier, est calculé dans le contrôle fini de la machine et ne nécessite aucun espace. Cela est possible en utilisant deux copies de M_h, une pour les chemins de poids positifs et l'autre pour les chemins de poids négatifs. L'état acceptant de M_f (resp. M_g) est l'état acceptant de M_h qui se situe dans la copie des poids positifs (resp. négatifs), l'autre copie de l'état acceptant devient un état non-terminal dont toutes les transitions entrantes ont poids 0.

Pour une entrée $x \in \{0,1\}^*$, on remarque que M_f (resp. M_g) calcule la somme des chemins de poids positifs (resp. négatif) dans l'arbre de calcul de M_h sur x. De plus, M_f et M_g sont des $\mathbb{N}$-MTA qui fonctionnent en temps polynomial ce qui implique que $f^{\mathbb{N}}_{M_f}$ et $f^{\mathbb{N}}_{M_g}$ sont des fonctions dans #P. Ainsi, puisque $h(x) = f^{\mathbb{N}}_{M_f}(x) - f^{\mathbb{N}}_{M_g}(x)$, on conclut que h est une fonction dans GAP. ■

1.4.5 Une définition de déterminisme pour la machine de Turing

Dans ce mémoire, nous aurons besoin d'une version dite déterministe de la machine de Turing. Une machine de Turing est déterministe (MTD) si pour toute configuration C_1, il existe au plus une configuration C_2 telle que C_1 mène à C_2 et que le poids de cette transition est 0 ou 1. Le poids zéro sert à rejeter l'entrée. On remarque ainsi qu'une MTD calcule toujours une fonction binaire et que la puissance expressive d'une MTD ne dépend pas du semi-anneau S puisque 0 et 1 agissent de la même façon peu importe S.

Définition 19.

- $TIME(t(n)) := \{f_M \mid M$ *est une* MTD *fonctionnant en temps* $O(t(n))\}$,
- $SPACE(s(n)) := \{f_M \mid M$ *est une* MTD *fonctionnant en espace* $O(s(n))\}$.
- $\mathrm{L} = SPACE(log(n))$,
- $\mathrm{P} = \bigcup_{k \in \mathbb{N}} TIME(n^k)$.

En fait, L et P sont des classes de langages mais la correspondance entre les fonctions binaires et les langages nous permet de les considérer comme des classes de fonctions. Suivant notre définition, on dira qu'un langage A est dans L (resp. P) si et seulement si χ_A est dans L (resp. P).

Pour une machine de Turing déterministe, il est d'usage de définir le calcul d'une fonction $f : \{0,1\}^* \to \{0,1\}^*$ comme suit :

Définition 20. *Soit M une machine de Turing, on dit que M calcule $f : \{0,1\}^* \to \{0,1\}^*$ si et seulement si M s'arrête avec exactement $f(x)$ sur son ruban lorsque l'entrée de M est $x \in \{0,1\}^*$.*

Afin de permettre à une machine de Turing déterministe de calculer une fonction f en utilisant un espace moindre que la taille de $f(x)$, on ajoute un ruban de sortie en écriture seulement. Seul le nombre de cellules utilisées sur le ruban de travail est considéré pour déterminer l'espace nécessaire au calcul de f.

Cette définition permet, par exemple, de calculer la somme de deux entiers encodés en binaire, $\langle a, b \rangle$, en utilisant un espace logarithmique alors que l'encodage binaire de cette somme, $\langle a + b \rangle$, est de taille linéaire par rapport à la taille de l'entrée.

1.5 Note sur l'uniformité du calcul

Dans la présente section, nous présentons la notion d'uniformité qui permettra de comparer la machine de Turing aux deux types de programmes. Nous utiliserons le nom « programme » afin de désigner à la fois les programmes de génération et les programmes de branchement algébrique et ce lorsque les explications s'appliqueront aux deux types de programmes.

Une différence fondamentale entre la machine de Turing algébrique et les programmes est qu'une machine de Turing algébrique calcule une fonction $f : \{0,1\}^* \to \{0,1\}$ alors qu'un programme calcule une fonction $f : \{0,1\}^n \to \{0,1\}$ pour $n \in \mathbb{N}$.

Si nous voulons comparer la machine de Turing algébrique aux autres modèles, nous devons donc considérer une famille de programmes, $(P_n)_{n\in\mathbb{N}}$, telle que P_n calcule la restriction, notée f_n, de f aux entrées de longueur n. Alors que la machine de Turing algébrique induit un même algorithme pour toute entrée sans égard à sa longueur, il n'y a pas nécessairement de connexion entre le programme P_n et le programme P_{n+1} d'une famille.

Pour cette raison, la machine de Turing algébrique est un modèle de calcul dit uniforme alors que les programmes de génération et les programmes de branchement algébrique sont des modèles de calcul dits non-uniformes. On peut toutefois définir plusieurs types d'uniformité pour une famille de programmes.

Définition 21. *Soit C une classe de fonctions et $P = (P_n)_{n\in\mathbb{N}}$ une famille de programmes, on dit que P est C-uniforme s'il existe une fonction*

$$DESCRIPTION^P : \{1\}^n \to \{0,1\}^* \in C$$

telle que $DESCRIPTION^P(1^n)$ est l'encodage binaire de P_n.

Nous utiliserons cet abus de notation.

Définition 22. *Une famille P de programmes est* L*-uniforme si la fonction $DESCRIPTION^P$ est calculée par une machine de Turing qui utilise un espace logarithmique.*

1.6 Relations entre les programmes de branchement algébrique et la machine de Turing algébrique

Il a déjà été mentionné qu'une des justifications théoriques de l'étude des programmes de branchement est le lien qui les relie aux machines de Turing. C'est Cobham, dans [Cob66], qui fut le premier à souligner que la taille d'un programme de branchement calculant une fonction n-aire est reliée à l'espace utilisé par une machine de Turing calculant la même fonction.

La construction de Christoph Meinel [Mei89] se généralise directement aux versions algébriques des deux modèles.

Proposition 23. *Une fonction f est dans S-$SPACE(log(n))$ si et seulement s'il existe une famille,* L*-uniforme, $(B_i)_{i\in\mathbb{N}}$ de programmes de branchement algébrique sur S telle que B_n calcule la restriction de f aux entrées de taille n et que la taille de B_n est polynomiale par rapport à n.*

1.7 Présentation et motivations du travail

Alors que le présent chapitre contient une présentation générale de plusieurs résultats présents dans la littérature récente de l'étude de la complexité, les chapitres à venir contiennent surtout des observations, définitions et résultats inédits.

Dans le prochain chapitre, nous étudierons les programmes de génération sur les semi-anneaux. Cette étude semble appropriée, car elle ne peut être directement déduite

des résultats connus pour les programmes de génération sur un corps. En effet, la notion d'inverse multiplicatif et de rang d'une matrice n'existe pas dans un semi-anneau. Nous devrons donc développer d'autres techniques afin de caractériser les programmes sur cette structure. Nous adapterons légèrement le modèle aux besoins de la structure. Nous analyserons ensuite la puissance expressive des programmes de génération sur un treillis borné et distributif ainsi que sur les naturels. Le chapitre sera terminé par une analyse du problème d'évaluation d'un programme de génération. Ce problème consiste à déterminer si un programme de génération sur un semi-anneau S accepte une entrée $x \in \{0,1\}^n$. En excluant la présentation des treillis bornés et distributifs, toutes les propositions et théorèmes de ce chapitre sont inédits. Il va de soit que certains résultats peuvent être reformulés en d'autres termes et contextes mais je n'ai trouvé aucun article traitant des programmes de génération sur un semi-anneau et j'ai tenté d'éliminer les résultats découlant directement de d'autres théories comme l'étude des équations diophantiennes.

Dans le dernier chapitre, nous aborderons l'étude des machines de Turing algébriques. Les premières sections de ce chapitre seront consacrées à l'adaptation de plusieurs résultats classiques à la machine algébrique. En particulier, on adaptera le théorème de Savitch qui stipule qu'une machine de Turing non-déterministe peut être efficacement simulée par une machine de Turing déterministe. Il s'agit bien ici d'adaptation dans le sens que les preuves de plusieurs propositions et théorèmes de ce chapitre sont inspirées des preuves classiques enseignées dans le cadre d'un cours gradué sur la théorie du calcul. Toutefois, j'estime que ce chapitre est fort utile en ce sens que l'adaptation ne fut pas toujours triviale. Le chapitre sera conclu par l'introduction de la machine de Turing algébrique alternante, un concept jamais étudié avant ce mémoire.

Chapitre 2

Programmes de génération sur les semi-anneaux

Il est naturel de vouloir généraliser les programmes de génération à des structures moins strictes que les corps. Dans le présent mémoire, nous avons choisi d'étudier les programmes de génération sur les semi-anneaux. Mentionnons qu'un corps est également un semi-anneau.

Puisque les semi-anneaux sont moins structurés que les corps, il devient impossible d'obtenir des résultats généraux s'appliquant à tous les semi-anneaux. Ainsi, dans ce chapitre, nous étudierons trois familles de semi-anneaux : les treillis bornés et distributifs, les nombres naturels et les graphes.

La première section de ce chapitre sera consacrée à l'adaptation du modèle pour les semi-anneaux. Nous analyserons, en seconde section, la puissance expressive des programmes de génération sur les treillis bornés et distributifs et sur les nombres naturels. Nous terminerons ce chapitre en analysant la complexité du problème d'évaluation d'un programme de génération en fonction du semi-anneau sur lequel il est défini.

2.1 Sémantique du modèle

Bien que la définition d'un programme de génération sur un semi-anneau soit la même que celle d'un programme de génération sur un corps, abstraction faite de la structure, il faut toutefois être vigilant pour ce qui est de la définition de calcul par un

tel programme.

La notion de combinaison linéaire n'étant pas très bien définie pour les semi-anneaux, il est utile d'en donner une définition que nous utiliserons par la suite.

Proposition 24. *Soit $k \in \mathbb{N}$, si $(S, +, \times)$ est un semi-anneau alors $(S^k, +^k, \times^k)$, où $+^k$ et $\times^k$ sont définis composante par composante, est un semi-anneau avec 0^k pour neutre additif et 1^k pour neutre multiplicatif.*

On peut alors considérer l'action $\otimes$ de S sur S^k définie comme suit :

Définition 25. *Soit $(S, +, \times)$ un semi-anneau, $s \in S$ et $(s_1, s_2, \ldots, s_k) \in S^k$:*

$$s \otimes (s_1, s_2, \ldots, s_k) = (s \times s_1, s \times s_2, \ldots, s \times s_k) \in S^k.$$

Soit $M \in S^{k \times m}$, on remarque que les colonnes de M sont éléments de S^k. Il est possible de redéfinir la notion de combinaison linéaire des colonnes de M comme suit.

Définition 26. *Soit $(S, +, \times)$ un semi-anneau, $M \in S^{k \times m}$ et $s = (s_1, s_2, \ldots, s_k) \in S^k$. On dit que s est une combinaison linéaire, sur S, des colonnes de M si et seulement s'il existe $y = (y_1, y_2, \ldots, y_m) \in S^m$ tel que :*

$$s = \sum_{i=1}^{m} y_i \otimes M_i.$$

Comme pour les corps, on utilise la notation $s \in \langle M \rangle$ pour signifier que l'élément s est une combinaison linéaire des colonnes de M. La définition de la fonction, $f_{\hat{M}}$, calculée par un programme de génération sur un semi-anneau est donc la même que pour les corps.

Définition 27. *Soit S un semi-anneau, $\hat{M} = (M, \mu, \tau)$ un programme de génération sur S et $x \in \{0,1\}^n$ où $n \in \mathbb{N}$, on définit $f_{\hat{M}} : \{0,1\}^n \to \{0,1\}$, la fonction binaire calculée par $\hat{M}$, par*

$$f_{\hat{M}}(x) = 0 \Leftrightarrow \tau \notin \langle M^x \rangle.$$

2.2 La puissance expressive

La puissance expressive d'un modèle de calcul, limité en ressource, est l'ensemble des fonctions que ce dernier peut calculer. À titre d'exemple, on a démontré dans le premier

chapitre que les programmes de génération de taille polynomiale sur $GF(2)$ capturent exactement la classe $\oplus$L, c'est-à-dire, la classe des langages décidés par une machine algébrique sur $GF(2)$ qui utilise un espace logarithmique.

Connaître la puissance expressive d'un modèle de calcul permet de mieux comprendre ce dernier. Par exemple, démontrer qu'un langage $A \in$ P ne peut être calculé par un programme de génération de taille polynomiale sur $GF(2)$ est équivalent à démontrer que $\oplus$L $\neq$ P. On peut donc s'attendre à ce que trouver un tel langage soit difficile.

Dans cette section, nous étudierons la puissance expressive des programmes de génération sur quelques semi-anneaux.

2.2.1 Les treillis bornés et distributifs

La théorie générale

Il est bien connu que les treillis bornés et distributifs (TBD) sont également des semi-anneaux. Nous allons utiliser les propriétés des TBD afin de démontrer que la puissance expressive d'un programme de génération sur un TBD est liée au calcul de suprémum et d'infimum sur ce dernier.

Le livre de Burris *et al.* [BS81] sert de référence pour les définitions et propositions suivantes. Ce livre, en version électronique, est disponible, via internet, à l'adresse :

« `http://www.math.uwaterloo.ca/%7Esnburris/htdocs/ualg.html` ».

Définition 28. *Un ensemble E muni de deux opérations binaires internes, $\vee$ et $\wedge$, est un treillis si les opérations sont commutatives, associatives et si*

- $\forall_{a \in E} \;\; a \vee a = a \wedge a = a$ *(idempotence),*
- $\forall_{a,b \in E} \;\; a \vee (a \wedge b) = a \wedge (a \vee b) = a$ *(absorption).*

Pour la seconde définition de treillis, nous aurons besoin de la notion d'infimum et de suprémum. Soit $\leq$ un ordre partiel sur un ensemble E, alors $\alpha \in E$ est un minorant (resp. majorant) de $A \subseteq E$ si et seulement si $\alpha \leq x$ (resp. $x \leq \alpha$) pour tout élément x de A.

Définition 29. *Soit E un ensemble et $A \subseteq E$, alors $\alpha \in E$ est l'infimum (resp. suprémum) de A si :*

1. *α est un minorant (resp. majorant) de A,*

2. *$\forall \beta \in E$, si β est un minorant (resp. majorant) de A alors $\beta \leq \alpha$ (resp. $\alpha \leq \beta$).*

Définition 30. *Un ensemble E muni d'un ordre partiel, $\leq$, est un treillis si et seulement si l'infimum et le suprémum de deux éléments de E existent toujours.*

L'équivalence entre les deux définitions est démontrée constructivement dans [BS81]. Nous nous contentons de décrire les constructions car il est utile de pouvoir utiliser les deux définitions parallèlement.

Définition 31. *Soit $\mathbb{T}$ un ensemble :*

(A) Si $\mathbb{T}$ est un treillis selon la définition 28 alors on définit l'ordre partiel sur $\mathbb{T}$ par :

$$\forall\ a, b \in \mathbb{T} \quad a \leq b \Leftrightarrow a = a \wedge b,$$

(B) Si $\mathbb{T}$ est un treillis selon la définition 30 alors on définit le suprémum et l'infimum de deux éléments de $\mathbb{T}$ par :

$$\forall\ a, b \in \mathbb{T} \quad a \vee b = sup\{a, b\} \quad et \quad a \wedge b = inf\{a, b\}.$$

Voici quelques exemples de treillis :

- $\mathbb{B} = (\{0, 1\}, \vee, \wedge)$,
- $(\mathbb{N}, \texttt{min}, \texttt{max})$,
- $(\wp(E), \subseteq)$, pour un ensemble E,
- $(\mathbb{N}, \leq)$,
- $(\mathbb{N}^0, \texttt{PGCD}, \texttt{PPCM})$
- $\mathbb{T}^k = (\mathbb{T}^k, \vee, \wedge)$, pour un treillis $\mathbb{T}$.

Il est facile de vérifier que $(\mathbb{N}, \texttt{min}, \texttt{max})$ et $(\mathbb{N}, \leq)$ sont en fait, selon nos constructions, le même treillis. Les opérations du treillis $\mathbb{T}^k$ sont définies composante par composante.

Définition 32. *Un treillis $\mathbb{T}$ est distributif si et seulement si :*

$$\forall\ a, b, c \in \mathbb{T} \quad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

On remarque que la commutativité des opérations d'un treillis permet de conclure que si un treillis est distributif alors

$$\forall\ a, b, c \in \mathbb{T} \quad (a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c).$$

Définition 33. *Un treillis $\mathbb{T}$ est borné si et seulement si le plus grand élément et le plus petit élément de $\mathbb{T}$ existent.*

Rappelons que la borne supérieure et la borne inférieure sont toujours uniques. En effet, si $a, b \in \mathbb{T}$ sont deux plus grands éléments de $\mathbb{T}$ alors $a = a \wedge b = b \wedge a = b$. Le raisonnement est analogue pour l'unicité du plus petit élément.

À titre d'exemple, les treillis $\mathbb{B}$ et $\wp(E)$ sont bornés et distributifs alors que les treillis $\mathbb{N}$ et $\mathbb{N}^0$ ne sont pas bornés supérieurement.

Proposition 34. *Un treillis borné et distributif est un semi-anneau.*

La preuve qui suit est un exemple patent de l'utilisation simultanée des deux définitions de treillis. En effet, la définition même de TBD suppose la coexistence de la notion d'ordre partiel et de deux opérations pour le treillis.

Démonstration. Soit $\mathbb{T} = (E, \vee, \wedge, 0, 1)$ un treillis borné et distributif avec 0 comme plus petit élément et 1 comme plus grand élément. On a déjà que les opérations sont associatives et commutatives par la définition de treillis. L'élément 0 est le neutre pour $\vee$ et l'élément 1 est le neutre pour $\wedge$. De plus, a est absorbant pour $\wedge$ pour tout $a \in \mathbb{T}$ et donc en particulier pour 0. ■

Il est maintenant possible de développer une argumentation s'appliquant à tous les treillis bornés et distributifs. La proposition suivante, simple en soi, est à la base de l'analyse qui suit :

Proposition 35. *Soit $\mathbb{T}$ un* TBD *et $a, b, c, t \in \mathbb{T}$. Alors*

$$a \wedge c \leq t \text{ et } b \wedge c \leq t \Leftrightarrow (a \vee b) \wedge c \leq t.$$

Démonstration. Soit $x, y, t \in \mathbb{T}$, on sait que si $x \leq t$ et $y \leq t$ alors $x \vee y \leq t$. En effet, puisque t est un majorant pour $\{x, y\}$ alors $(x \vee y) = sup\{x, y\} \leq t$. De plus, puisque $\mathbb{T}$ est un TBD, on sait que $(a \wedge c) \vee (b \wedge c) = (a \vee b) \wedge c$ pour tout $a, b, c \in \mathbb{T}$. On conclut la proposition 35 en posant $x = a \wedge c$ et $y = b \wedge c$. ■

Soit $M \in \mathbb{T}^{k \times m}$ la matrice d'un programme de génération sur un treillis $\mathbb{T}$. Pour une entrée $x \in \{0, 1\}^n$, considérons M_i et M_j, deux colonnes de M^x. On sait que $M_i \vee M_j \leq \tau$

si et seulement si $M_i \leq \tau$ et $M_j \leq \tau$ puisque $\mathbb{T}^k$ est aussi un TBD. Ainsi, on ne peut utiliser une colonne M_i avec une composante $M_i[l] \not\leq \tau[l]$ à moins de la multiplier ($\otimes$) par une constante bien choisie. Cette observation se formalise par la proposition suivante.

Proposition 36. *Soit* $\mathbb{T}$ *un* TBD, $\hat{M} = (M, \mu, \tau)$ *un* $\mathbb{T}$-SP *et* $x \in \{0,1\}^n$ *une entrée pour* $\hat{M}$ *alors* x *est accepté par* $\hat{M}$ *si et seulement si* $\tau = \bigvee_{M_i \in M^x} (\alpha_i \otimes M_i)$, *où :*

$$\alpha_i = sup\ \{\beta \in \mathbb{T} \mid \beta \otimes M_i \leq \tau\}.$$

Démonstration. Seule la direction « seulement si » est à prouver. Supposons que

$$\tau \neq \bigvee_{M_i \in M^x} (\alpha_i \otimes M_i).$$

On sait, par la proposition 35, que

$$\alpha_i \otimes M_i \leq \tau,$$

pour tout i, on en déduit que

$$\bigvee_{M_i \in M^x} (\alpha_i \otimes M_i) \lneq \tau.$$

Sachant que pour tout $\beta \in \mathbb{T}$ on a

$$\beta \otimes M_i \leq \tau \Rightarrow \beta \leq \alpha_i,$$

on peut déduire que

$$\beta \otimes M_i \leq \tau \Rightarrow \beta \otimes M_i \leq \alpha_i \otimes M_i.$$

Ainsi, toute famille de coefficients (β_i) est telle que, si

$$\bigvee_{M_i \in M^x} (\beta_i \otimes M_i) \leq \tau,$$

alors

$$\bigvee_{M_i \in M^x} (\beta_i \otimes M_i) \leq \bigvee_{M_i \in M^x} (\alpha_i \otimes M_i) \lneq \tau.$$

M^x ne peut donc pas générer τ, ce qui termine la preuve. ■

La proposition précédente devient très utile dans certains cas. Par exemple, pour un TBD fini, on peut trouver les α_i en temps linéaire par rapport au nombre de lignes de la matrice M. En effet, pour chaque élément $\beta \in \mathbb{T}$, on calcule $\beta \otimes M_i$ et on compare avec τ. On calcule ensuite le suprémum des β tels que $\beta \otimes M_i \leq \tau$.

Si le TBD est une chaîne, c'est-à-dire un ordre total, on obtient le corollaire suivant.

Corollaire 37. *Soit $\mathbb{T}$, une chaîne, $\hat{M} = (M, \mu, \tau)$ un $\mathbb{T}$-SP et $x \in \{0,1\}^n$ une entrée pour $\hat{M}$ alors x est accepté par $\hat{M}$ si et seulement si $\tau = \bigvee_{M_i \in M^x} (\alpha'_i \otimes M_i)$ où les $\alpha'_i \in \mathbb{T}$ sont donnés par*

$$\alpha'_i = \begin{cases} 1 & \text{si } M_i \leq \tau \\ \texttt{MIN}\{\tau[j] \mid M_i[j] \not\leq \tau[j]\} & \text{sinon.} \end{cases} \tag{2.1}$$

Démonstration. Soit α_i tel que défini dans la proposition précédante, il suffit de montrer que $\alpha'_i = \alpha_i$ pour tout i. Pour $M_i \leq \tau$, on sait que $1 \otimes M_i \leq \tau$, on déduit que $\alpha'_i = 1 = \alpha_i$. En effet, 1 est le plus grand élément de $\mathbb{T}$ et donc le suprémum de l'ensemble $\{\beta \in \mathbb{T} \mid \beta \otimes M_i \leq \tau\}$. Pour $M_i \not\leq \tau$, on définit $J(i) = \{j \mid M_i[j] \not\leq \tau[j]\}$. On observe que $\beta \otimes M_i \leq \tau$ si et seulement si $\beta \leq \tau[j]$ pour tout $j \in J(i)$. Le plus grand des $\beta \in \mathbb{T}$ respectant cette condition est $\texttt{MIN}\{\tau[j] \mid M_i[j] \not\leq \tau[j]\}$. On conclut que $\alpha'_i = \alpha_i$. ■

Un exemple : le semi-anneau des booléens

Le semi-anneau des booléens, défini par le triplet $\mathbb{B} = (\{0,1\}, \vee, \wedge)$, est un exemple célèbre de TBD. En fait, le semi-anneau des booléens est également une chaîne. Nous allons montrer que la puissance expressive des programmes de génération sur $\mathbb{B}$ est exactement la même que celle des formules en forme normale conjonctive.

Voici un rappel de la définition d'une formule en forme normale conjonctive.

Définition 38.

1. *Une formule de n variables est en forme normale conjonctive (FNC) si elle est de la forme $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ avec les C_i, appelées clauses, de la forme $l_{i,1} \vee l_{i,2} \vee \cdots \vee l_{i,j_i}$ où $l_{i,j} \in X_n$ pour tout i, j.*

2. *La taille d'une formule F en forme normale conjonctive est le nombre de clauses dans F.*

La proposition suivante permettra de simplifier l'analyse subséquente. En effet, elle permet de se limiter à l'étude des programmes de génération ayant un vecteur cible de la forme 1^k pour $k \in \mathbb{N}$.

Proposition 39. *Si un $\mathbb{B}$-SP est de taille minimale pour une fonction booléenne n-aire f alors son vecteur cible est de la forme 1^k, où k est le nombre de lignes de la matrice du programme.*

Démonstration. Soit $\hat{M} = (M, \mu, \tau)$ un $\mathbb{B}$-SP de taille minimale pour f. Supposons que $\tau \neq 1^k$. Soit j un indice tel que $\tau[j] = 0$. Puisque $\mathbb{B}$ est un TBD, le corollaire 37 s'applique. Ainsi, pour tout $x \in \{0,1\}^n$, x est acceptée par $\hat{M}$ si et seulement si, pour chaque colonne M_i de M^x avec $M_i[j] = 1$, on a $\alpha_i = 0$ dans $\bigvee_{M_i \in M^x} (\alpha_i \otimes M_i)$. Autrement dit, ces colonnes ne sont pas utiles pour calculer f. Une fois ces colonnes éliminées, la ligne j de M ne contient que des 0, on peut donc l'éliminer. Il en résulte également que la composante j de τ est éliminée. On a donc obtenu un nouveau programme de génération calculant f de taille inférieure à $\hat{M}$ contredisant la minimalité de ce dernier.

■

Soit $\hat{M}$ un programme de génération sur $\mathbb{B}$ avec $\tau = 1^k$, le corollaire 37 devient : $x \in \{0,1\}^n$ est acceptée par $\hat{M}$ si et seulement si $\bigvee_{M_i \in M^x} M_i = 1^k$. Cela implique que $\tau \in \langle M^x \rangle$ si et seulement si, pour chaque indice j, il existe une colonne M_i de M^x telle que $M_i[j] = 1$. Plus formellement,

$$f_{\hat{M}}(x) = 1 \Leftrightarrow \forall j \in [k]\ \exists M_i \in M^x\ M_i[j] = 1. \tag{2.2}$$

On en déduit qu'il est inutile d'avoir plus d'une colonne de M étiquetée par un même littéral car on peut fusionner de telles colonnes avec l'opération $\vee$. Ainsi, le nombre de colonnes nécessaires pour un programme de génération sur $\mathbb{B}$ calculant une fonction $f : \{0,1\}^n \to \{0,1\}$ est au plus le nombre de littéraux soit : $2 \cdot n$.

Cette observation justifie qu'il faut considérer le nombre de lignes d'un programme afin d'obtenir une bonne mesure de complexité pour les programmes de génération sur

les semi-anneaux. En effet, pour le semi-anneau des booléens, les deux autres paramètres, soient le nombre de colonnes et la taille d'encodage d'une entrée de M, ne dépendent que de l'arité et non de la fonction.

De plus, en travaillant sur $\mathbb{B}^k$ au lieu de $\mathbb{B}$, il est possible de calculer une fonction par un programme de génération n'utilisant qu'une ligne au lieu de k lignes. Ainsi, toutes les fonctions calculables par un programme de génération sur $\mathbb{B}$ utilisant k lignes sont également calculables par un programme de génération sur $\mathbb{B}^k$ n'utilisant qu'une ligne. La taille des deux programmes sera toutefois la même puisque l'encodage d'une cellule du programme sur $\mathbb{B}^k$ sera k fois plus grande que pour une cellule du programme sur $\mathbb{B}$ puisque qu'un élément de $\mathbb{B}^k$ est encodé par l'encodage de k éléments de $\mathbb{B}$.

Ainsi, la taille d'encodage d'un élément de M et le nombre de lignes de M sont à considérer afin d'obtenir une bonne mesure de taille pour les programmes de génération sur $\mathbb{B}^k$ et $\mathbb{B}$ respectivement.

Tout est maintenant en place afin de démontrer le résultat principal de cette section.

Proposition 40. *Soit $f : \{0,1\}^n \to \{0,1\}$ une fonction binaire, alors il existe un $\mathbb{B}$-SP de k lignes calculant f si et seulement s'il existe une formule FNC de taille au plus k calculant f.*

Démonstration. Soit $\hat{M} = (M, \mu, \tau)$, un $\mathbb{B}$-SP calculant f, on construit la formule FNC suivante :

1. Pour chaque ligne l_j de M, on construit une clause C_j.
2. Les littéraux de C_j sont les étiquettes des colonnes de M dont la composante j est 1.

Soit F, une formule FNC, on construit la matrice suivante :

1. Pour chaque clause C_j de F on construit une ligne l_j.
2. On construit $2 \cdot n$ colonnes étiquetées par les $2 \cdot n$ littéraux.
3. La position (i, j) de la matrice M est 1 si et seulement si $\mu(M_j)$ est dans C_i.

La validité des constructions est directement vérifiée en utilisant l'équation 2.2. ■

Une généralisation

On peut se demander s'il est possible d'augmenter la puissance expressive en utilisant une plus grande chaîne que les booléens. La réponse est négative. Le résultat suivant démontre que la proposition 40 s'adapte à toutes les chaînes.

Proposition 41. *Soit $\mathbb{T}$ une chaîne et f, une fonction binaire d'arité $n \in \mathbb{N}$ alors il existe un $\mathbb{T}$-SP de k lignes calculant f si et seulement s'il existe une formule FNC de taille au plus k calculant f.*

Démonstration. Soit $\hat{M} = (M, \mu, \tau)$, un $\mathbb{T}$-SP minimal calculant f. On remarque qu'en utilisant une preuve similaire à celle de la proposition 39, il est possible de démontrer que le vecteur cible τ n'a pas de composante nulle. Soit α_i le coefficient de M_i donné par la proposition 36, on peut donc construire la formule FNC suivante.

1. Pour chaque ligne l_j de M' on construit une clause C_j.
2. Les littéraux de C_j sont les étiquettes des colonnes M_i telles que $\alpha_i \otimes M_i[j] = \tau[j]$.

Puisque $\mathbb{T}$ est un ordre total, on observe que $f_{\hat{M}}(x) = 1$ si et seulement si

$$\texttt{MAX}\{\alpha_i \otimes M_i[j] \mid j \in [k]\} = \tau[j],$$

où la somme (MAX) est prise sur toutes les colonnes M_i de M^x. Cela n'est donc possible que s'il existe une colonne M_i telle que $\alpha_i \otimes M_i[j] = \tau[j]$ ce qui justifie la deuxième étape de la construction.
La construction inverse est identique à la seconde partie de la preuve donnée pour la proposition 40.

■

2.2.2 Le semi-anneau des naturels

Le semi-anneau des nombres naturels est le triplet habituel $\mathbb{N} = (\mathbb{N}, +, \cdot)$. Nous allons démontrer que les programmes de génération de taille polynomiale sur $\mathbb{N}$ permettent de décider tous les problèmes de la classe de complexité NL.

Pour ce faire, on utilise le fait qu'une machine de Turing non-déterministe M utilisant un espace logarithmique peut être représentée par une famille de graphes orientés et

étiquetés, $(G_n)_{n\in\mathbb{N}}$, de taille polynomiale. Le graphe G_n est le graphe de toutes les configurations de M possibles pour une entrée de longueur n. Dans G_n, il y a un arc étiqueté x_j^σ reliant le sommet C_a au sommet C_b si et seulement si j est la position de la tête de lecture de M pour la configuration C_a et qu'il existe une transition de C_a à C_b lisant le caractère σ.

La taille du graphe G_n est le nombre de configurations possibles, ce qui est dans $O(n \cdot 2^{O(lgn)})$.

Ainsi, une entrée $x \in \{0,1\}^n$ est acceptée par M si et seulement s'il existe un chemin de C_0 à C_f, dans G_n, consistant avec x.

Rappelons que les graphes G_n sont acycliques puisqu'on ne considère que les machines de Turing qui s'arrêtent sur toutes les branches de calculs.

Un graphe orienté $G = (V, E)$ est dit ordonné si pour tous les arcs $(a, b) \in E$, on a $a < b$. La matrice d'adjacence d'un graphe ordonné est donc triangulaire supérieure. La taille de l'encodage d'un tel graphe est la taille de sa matrice d'adjacence soit $|V|^2$.

Il est facile de modifier un graphe orienté (sans cycle) G de taille t en un graphe orienté et ordonné G' en utilisant t copies de G. Si a est un sommet de G, on note alors a_i le sommet correspondant à a dans la i-ème copie de G. Les arcs de G' sont exactement les arcs (a_i, b_{i+1}) tels que (a, b) est un arc de G.

Définition 42. *On définit le problème*

$$\texttt{GAP}^< = \{\langle G\rangle \mid G \text{ est un graphe orienté, ordonné qui possède un chemin de } 1 \text{ à } |G|\}.$$

Proposition 43. *Pour chaque $v \in \mathbb{N}$, il est possible de décider $\texttt{GAP}^<$ pour les graphes de v sommets par un programme de génération de taille $(v-1)\cdot\binom{v}{2}\cdot 1$.*

Démonstration. Soit $v \in \mathbb{N}$, considérons le programme $\hat{M} = (M, \mu, \tau)$ suivant. La matrice M possède $v-1$ lignes. Les colonnes de M sont étiquetées dans l'ordre suivant :

$$x_{1,2}, x_{1,3}, \ldots, x_{1,v}, x_{2,3}, \ldots, x_{v-1,v}.$$

La colonne étiquetée par $x_{i,j}$ est $(1^{j-1}0^{(v-j)})^T$ si $i = 1$ et $(0^{i-1}1^{(j-i)}0^{v-j})^T$ sinon. Le vecteur cible τ est tout simplement 1^{v-1}. Pour $v = 4$, on a $\tau_4 = 111$ et M_4 donnée à la figure 2.1.

x_{12}	x_{13}	x_{14}	x_{23}	x_{24}	x_{34}
1	1	1	0	0	0
0	1	1	1	1	0
0	0	1	0	1	1

FIG. 2.1 – La matrice M_4 avec les étiquettes des colonnes.

Nous montrons maintenant que $\hat{M}$ calcule bien $\texttt{GAP}^{<}$.

Prenons $G \in \texttt{GAP}^{<}$, alors il existe un chemin $P = (a_i, b_i)_{i\in[k]}$ avec $a_1 = 1$ et $b_k = v$. On sait de plus que $a_i < b_i = a_{i+1}$. La colonne étiquetée par x_{a_i,b_i} contient des 1 exactement aux positions a_i à $b_i - 1$. Ainsi, soit M_P l'ensemble des colonnes de M_G étiquetées par les arcs de P, on conclut que $\sum_{r\in M_P} r = 1^{v-1} = \tau$.

Prenons maintenant $G \notin \texttt{GAP}^{<}$, supposons qu'il existe une combinaison linéaire des colonnes de M_G permettant de générer τ. Puisque $\tau = 1^{v-1}$, il est garanti que tous les coefficients de cette combinaison sont des 1. On sait également que les colonnes de la combinaison sont disjointes deux-à-deux car sinon la somme aurait une composante plus grande que 1. Soit $(r_i)_{i\in[k]}$, la suite des colonnes de M_G utilisées dans cette combinaison linéaire, ordonnée selon leurs étiquettes. Nous montrerons que $P = (a_i, b_i)_{i\in[k]}$, où $\mu(r_i) = x_{a_i,b_i}$ est un chemin de 1 à v dans le graphe G.

Il est clair que $a_1 = 1$ car la première composante de $\sum r_i = S$ est 1. De plus, on remarque que $b_k = v$ car la dernière composante de S est 1. Il ne nous reste donc qu'à démontrer que $a_i < b_i = a_{i+1}$ pour tous les i plus petits que k. L'inégalité est garantie par la construction de $\hat{M}$. On démontre l'égalité en notant que si $b_i \neq a_{i+1}$ alors $S(a_{i+1}) = 0$ ce qui contredit $S = \sum r_i = \tau = 1^{v-1}$.

Nous avons donc une contradiction car P est un chemin de 1 à v dans G or on a supposé qu'il n'y en avait pas. On conclut donc que τ ne peut pas être généré par M_G. On a donc démontré que $G \in \texttt{GAP}^{<}$ si et seulement si τ est généré par M_G. De plus, la taille de $\hat{M}$ est bien $(v-1)\cdot\binom{v}{2}\cdot 1 \in O(|V|^3)$.

■

Il devient maintenant possible de justifier que tous les langages de NL sont décidés par une famille (L-uniforme) de programmes de génération sur $\mathbb{N}$ de taille polynomiale.

Corollaire 44. *Si $A \in$ NL alors il existe une famille, $(\hat{M})_{n\in\mathbb{N}}$, de programmes de*

génération sur $\mathbb{N}$ *qui calcule* χ_A *et telle que* $\hat{M}_n$ *est de taille polynomiale par rapport à* n.

Démonstration. Soit N une machine non-déterministe qui décide A en utilisant un espace logarithmique. Considérons $(G_n)_{n\in\mathbb{N}}$ la famille de graphes orientés et ordonnés associée à N. On sait que la taille t_n du graphe G_n est polynomialement reliée à n. Soit $\hat{M}_{t_n}$ le programme de génération, donné dans la proposition 43, qui calcule $\texttt{GAP}^<$ pour les graphes de t_n sommets. Il suffira de modifier quelque peu $\hat{M}_{t_n}$ afin de calculer $\chi_A^{t_n}$. En effet, fixons $n \in \mathbb{N}$, simplifions l'écriture en identifiant G à G_n et t à t_n. On a donc un graphe G de t sommets dont les arcs sont étiquetés, par un littéral sur n variables, selon la fonction de transition de M.

Considérons le programme $\hat{M}_t^G = (M_t^G, \mu_t^G, \tau_t)$ où M_t^G ne contient que les colonnes étiquetées par un arc présent dans G et μ_t^G est définie comme suit. Pour chaque colonne M_i de M_t^G, la fonction μ_t^G associe l'étiquette de l'arc $\mu_t(i)$ dans le graphe G. Pour $x \in \{0,1\}^n$, la matrice $(M_t^G)^x$ contient donc exactement les colonnes de M_t qui sont associées aux arcs de G consistants avec x. Ainsi, $\tau \in \langle (M_t^G)^x \rangle$ si et seulement si G_n contient un chemin du sommet C_0 au sommet C_f consistant avec x. On termine la preuve en remarquant que cela est possible si et seulement si M accepte x. ■

Pour une machine de Turing N fixée, il est possible de construire le graphe G_n en utilisant un espace logarithmique par rapport à une entrée 1^n où $n \in \mathbb{N}$. À partir d'un graphe G_n fixé, il est possible de construire le programme de génération $\hat{M}_t^G$ en utilisant un espace logarithmique par rapport à la taille de G_n. En combinant ces deux constructions, on obtient une famille de programmes de génération L-uniforme.

2.3 Le problème d'évaluation

L'analyse de la complexité du problème d'évaluation pour un modèle de calcul permet d'obtenir une borne supérieure sur sa puissance expressive. De plus, elle permet d'évaluer la valeur pratique d'un modèle de calcul. Un bon exemple est l'utilisation des automates finis déterministes (AFD) pour la recherche d'expressions dans un texte. En effet, on sait que la simulation du calcul d'un AFD par une machine de Turing est

linéaire, ce qui est très efficace. De plus, il est tout aussi efficace de construire un AFD qui trouve les occurrences d'une expression dans un texte.

Bien que cet aspect pratique soit moins évident pour les programmes de génération, l'étude du problème d'évaluation reste utile. En particulier, on démontre que le problème d'évaluation d'un programme de génération sur les naturels est NP-complet. Cela suggère un axe de recherche intéressant pour l'avenir. En effet, la borne inférieure, on sait qu'il est possible de calculer toutes les fonctions de NL, est très loin de la borne supérieure, l'évaluation est NP-complet.

Voici la définition formelle du problème d'évaluation d'un programme de génération.

Définition 45. *Problème d'évaluation d'un programme de génération sur S ($\mathtt{A}_S$) :*

Entrée : $\langle x, \hat{M} \rangle$, *un programme de génération* $\hat{M}$ *sur* S *et une entrée* $x \in \{0,1\}^n$.

Question : Est-ce que $\hat{M}$ *accepte* x *?*

Bien sûr, la complexité de ce problème varie selon le semi-anneau. Dans cette section, les booléens, les naturels et les graphes seront considérés.

2.3.1 Le semi-anneau des booléens

Le corollaire 37 indique que l'évaluation d'un programme de génération sur une chaîne $\mathbb{T}$ est au plus aussi difficile que le calcul du minimum sur ce dernier. En effet, on a démontré qu'une entrée $x \in \{0,1\}^n$ est acceptée par un $\mathbb{T}$-SP, $\hat{M} = (M, \mu, \tau)$, si et seulement si $\tau = \bigvee_{M_i \in M^x} (\alpha_i \otimes M_i)$, où les α_i sont donnés par :

$$\alpha_i = \begin{cases} 1 & \text{si } M_i \leq \tau \\ \texttt{MIN}\{\tau[j] \mid M_i[j] \not\leq \tau[j]\} & \text{sinon.} \end{cases} \tag{2.3}$$

Dans le cas des booléens, le minimum de plusieurs éléments est simplement la conjonction de ces derniers.

Il est important de noter que nous ne pouvons supposer que le vecteur cible est de la forme 1^k. En effet, le problème $\mathtt{A}_{\mathbb{B}}$ est défini pour tous les programmes de génération ce qui inclut les programmes non-minimaux.

La fonction suivante, qui utilise la notation de la proposition 36, sera très utile par la suite. La fonction, $g_{\hat{M}}$, est définie sur le domaine $\{0,1\}^n$ alors que la proposition 36

suppose un $x \in \{0,1\}^n$ fixé. Nous ajoutons donc un critère afin que, pour $x \in \{0,1\}^n$, la fonction ne considère que les colonnes de M^x. On sait que $\mu(M_j) = x_i^\sigma$ est un littéral associé à la j-ème colonne de la matrice M du programme $\hat{M}$. Il suffit de considérer les littéraux comme une projection afin d'atteindre notre but.

Définition 46. *Une projection $x_i^\sigma : \{0,1\}^n \to \{0,1\}$ est une fonction binaire qui, sur entrée $x \in \{0,1\}^n$, retourne la valeur 1 si et seulement si x satisfait le littéral x_i^σ.*

Dans la définition qui suit, le 0 et le 1 de $\{0,1\}$ sont identifiés au 0 et au 1 de $\mathbb{T}$.

Définition 47. *Pour un programme de génération $\hat{M} = (M, \mu, \tau)$ sur $\mathbb{T}$, on définit $g_{\hat{M}} : \{0,1\}^n \to \mathbb{T}$ par*

$$g_{\hat{M}}(x_1, x_2, \ldots, x_n) = \bigvee_{M_j \in M} ((\mu(M_j) \wedge \alpha_j) \otimes M_j).$$

Il est immédiat, par la proposition 36, que

$$f_{\hat{M}}(x_1, x_2, \ldots, x_n) = 0 \Leftrightarrow g_{\hat{M}}(x_1, x_2, \ldots, x_n) = \tau.$$

Ainsi, pour calculer $f_{\hat{M}}$, il suffit de calculer $g_{\hat{M}}$ et de comparer la valeur retournée avec la valeur de τ.

Dans ce qui suit, le corollaire 37 et la fonction $g_{\hat{M}}$ seront utilisés afin d'évaluer un programme de génération sur les booléens à l'aide de deux circuits de profondeur constante. En combinant ces circuits de profondeur constante, on démontre que $\mathtt{A}_{\mathbb{B}}$ est dans AC^0.

Rappelons que la classe AC^0 est la classe des fonctions binaires calculées par une famille de circuits booléens de profondeur constante, de taille polynomiale et dont les portes sont d'entrance non-bornée. La taille d'un circuit est le nombre de portes de ce dernier. La puissance expressive de AC^0 est très limitée. En effet, Furst *et al.* [FSS84] ont démontré que décider la parité de n valeurs binaires n'est pas possible avec de tels circuits.

Afin d'éviter toute confusion, il est important de déterminer un encodage «honnête» pour une entrée $\langle \hat{M} = (M, \mu, \tau), x \rangle$. Dans ce qui suit, $\langle EXPRESSION \rangle$ désigne l'encodage de $EXPRESSION$. De plus, on considérera μ comme un vecteur.

Définition 48. *Soit* $i, n \in \mathbb{N}$, $\sigma \in \{0,1\}$, $x \in \{0,1\}^n$, $M \in \mathbb{B}^{k \times m}$, $\mu \in (X_n)^m$ *et* $\tau \in \mathbb{B}^k$ *alors* $\langle x, (M, \mu, \tau)\rangle = \langle x\rangle\langle M\rangle\langle \mu\rangle\langle \tau\rangle$ *où :*

- *Pour les éléments de* $\{0,1\}$, $\langle 0\rangle = 0$, $\langle 1\rangle = 1$
- $\langle x\rangle = \langle x[1]\rangle\langle x[2]\rangle \cdots \langle x[n]\rangle$,
- $\langle x_i^\sigma\rangle = 1^i 0^{n-i}\langle\sigma\rangle$,
- $\langle \mu\rangle = \langle \mu[1]\rangle\langle \mu[2]\rangle \cdots \langle \mu[m]\rangle$,
- $\langle \tau\rangle = \langle \tau[1]\rangle\langle \tau[2]\rangle \cdots \langle \tau[k]\rangle$,
- $\langle M_i\rangle = \langle M_i[1]\rangle\langle M_i[2]\rangle \cdots \langle M_i[k]\rangle$, *où* M_i *est la* i*-ème colonne de* M,

Proposition 49. *Le langage* $\mathtt{A}_{\mathbb{B}}$ *est dans* AC^0.

Démonstration. Les circuits sont un modèle de calcul non-uniforme. Trois paramètres seront considérés pour indicer notre famille de circuits calculant $\mathtt{A}_{\mathbb{B}}$: la longueur de x, le nombre de colonnes de M et le nombre de composantes de τ.

Dans ce qui suit, nous considérons que $x \in \{0,1\}^n$, que M possède m colonnes et que $\tau \in \mathbb{B}^k$ pour $n, m, k \in \mathbb{N}$. Ainsi, la longueur de $\langle x, (M, \mu, \tau)\rangle$ est exactement $(n + k \cdot m + m \cdot 2n + k \cdot 1)$. Cet encodage nous permet de déterminer exactement la forme du circuit pour n, m, k fixés. Nous pouvons maintenant décrire les circuits qui seront combinés pour calculer $\mathtt{A}_{\mathbb{B}}$.

Le premier circuit sert à déterminer les colonnes de M^x. Cela est possible en profondeur 4.

On sait que la colonne i de M est dans M^x si et seulement si $\mu[i]$ est satisfait par l'entrée x. Fixons i et posons $\mu[i] = x_j^\sigma$, on peut déterminer la valeur de j en profondeur 2. Le premier niveau contient $n-1$ portes négations : soit une par symbole de $\langle \mu[i]\rangle$ sauf pour $\langle \mu[1]\rangle$. Au second niveau, on a n portes de type «ET». La l-ème porte «ET» calcule la conjonction des l premières composantes de $\langle \mu[i]\rangle$ et de la négation de la composante $l+1$ calculée au niveau 1. Ainsi, pour $\langle \mu[i]\rangle$, exactement une des portes du second niveau est vraie et il s'agit de la j-ème porte «ET».

En parallèle, on compare toutes les composante de x avec σ. Considérant que $x[i] = \sigma \Leftrightarrow (x[i] \wedge \sigma) \vee (\neg x[i] \wedge \neg\sigma)$, deux niveaux suffisent pour la comparaison. On a donc un circuit avec n portes de sortie de type «OU».

Au troisième niveau du circuit, on a n portes de type «ET» telles que la l-ème est alimentée par les l-ème sorties des deux sous-circuits précédents. Le niveau 4 du circuit

est simplement la disjonction des portes du niveau 3.

En combinant parallèlement m copies du premier circuit, on obtient un circuit de profondeur 4 avec m portes de sortie telles que la i-ème porte de sortie est vraie si et seulement si M_i est dans M^x.

Le second circuit sert à déterminer les colonnes «utiles», de M, afin générer τ. C'est-à-dire qu'on doit éliminer les colonnes de M avec une composante j plus grande que $\tau[j]$. On veut donc un circuit avec n portes de sortie tel que la l-ème porte est vraie si et seulement si $M_i \leq \tau$.

Le second circuit traite parallèlement toutes les colonnes de M. Voici la description du circuit pour une colonne. On a vu que $a \leq b \Leftrightarrow a = a \wedge b$. Vérifier qu'une composante j de M_i est plus petite ou égale à $\tau[j]$ revient donc à vérifier si $M_i[j] = M_i[j] \wedge \tau[j]$. Cela est possible en profondeur 2. On combine ensuite les k portes «OU» du second niveau par une porte « ET ».

On peut maintenant connecter les portes correspondantes du circuit 1 et du circuit 2 par une conjonction afin de déterminer les colonnes de M^x qui peuvent être utiles. On a donc un circuit de profondeur 5 avec m portes de sortie telles que la l-ème porte est vraie si et seulement si M_i est dans M^x et M_i est « utile ».

Notons U^x la matrice contenant uniquement les colonnes utiles de M^x. La dernière étape consiste à vérifier si la somme des colonnes de U^x est égale à τ. On fera le calcul parallèlement pour chaque composante $j \in [k]$ et on termine en faisant la conjonction des k résultats partiels.

Pour une composante j, on doit vérifier que $\bigvee_{M_i \in U^x} M_i[j] = \tau[j]$. Cela est possible comme suit. Pour chaque colonne M_i de M, on calcule si $M_i[j] = \tau[j]$ en deux niveaux. Le niveau 6 contient la conjonction de ce résultat avec la i-ème porte de sortie du niveau 5. On a donc m portes telles que la l-ème est vraie si et seulement si $M_l[j] = \tau[j]$ et M_l est une colonne de U^x. On fait ensuite la disjonction des m portes et on obtient un circuit de profondeur 7 qui retourne la valeur vrai si et seulement si $\bigvee_{M_i \in U^x} M_i[j] = \tau[j]$. En faisant cela parallèlement pour toutes les composantes, on obtient k portes de type « OU » au niveau 7 telles que la l-ème porte est satisfaite si et seulement si $\bigvee_{M_l \in U^x} M_l[j] = \tau[j]$.

On termine le circuit par une porte « ET » qui calcule la disjonction des portes du

septième niveau.

La profondeur du circuit est constante, il y a huit niveaux, et la taille est polynomiale par rapport à n, m et k. On conclut que le problème $\mathtt{A}_{\mathbb{B}}$ est dans AC^0. ■

2.3.2 Le semi-anneau des naturels

Dans cette section, nous montrons que le problème d'évaluation pour les programmes de génération sur les naturels est `NP-complet`. Pour ce faire, nous allons montrer que le problème `1in3m3SAT` se réduit au problème d'évaluation d'un $\mathbb{N}$-SP.

La notion de complétude pour la classe NP a été introduite par Cook [Coo71]. Bien que Cook n'ait pas explicitement parlé de complétude, il a démontré l'existence d'un problème tel que P = NP si et seulement si ce dernier est dans P. Pour comprendre la notion de complétude, il faut introduire celle de réduction.

Définition 50. *Soit A et B deux langages binaires. On dit que A se réduit à B, au sens multivoque ($A \leq_m B$), si et seulement s'il existe une fonction $f : \Sigma_1^* \to \Sigma_2^*$ calculable telle que :*

$$\forall\, \omega \in \Sigma_1^*,\ \omega \in A \Leftrightarrow f(\omega) \in B.$$

Si la fonction f est calculable en espace logarithmique, on dit que A se réduit logarithmiquement à B au sens multivoque ($A \leq_m^{log} B$).

Définition 51. *Un langage L sur un alphabet Σ est* `NP-complet` *selon $\leq_m^{log}$ si :*

1. *L est élément de* NP,
2. $\forall\, A \in \mathrm{NP},\ A \leq_m^{log} L$.

Nous utiliserons `NP-complet` pour signifier `NP-complet` selon la réduction multivoque en espace logarithmique.

La complétude de plusieurs problèmes, pour la classe NP, a depuis été démontrée. Nous vous référons au livre de Garey et Johnson [GJ90] pour obtenir une liste non-exhaustive de tels problèmes. L'avantage de connaître plusieurs problèmes `NP-complet` est illustré dans la proposition suivante.

Proposition 52. *Si A est* `NP-complet`, *$B \in$ NP et $A \leq_m^{log} B$ alors B est* `NP-complet`.

septième niveau.

La profondeur du circuit est constante, il y a huit niveaux, et la taille est polynomiale par rapport à n, m et k. On conclut que le problème $\mathtt{A}_{\mathbb{B}}$ est dans AC^0. ■

2.3.2 Le semi-anneau des naturels

Dans cette section, nous montrons que le problème d'évaluation pour les programmes de génération sur les naturels est `NP-complet`. Pour ce faire, nous allons montrer que le problème `1in3m3SAT` se réduit au problème d'évaluation d'un $\mathbb{N}$-`SP`.

La notion de complétude pour la classe NP a été introduite par Cook [Coo71]. Bien que Cook n'ait pas explicitement parlé de complétude, il a démontré l'existence d'un problème tel que P = NP si et seulement si ce dernier est dans P. Pour comprendre la notion de complétude, il faut introduire celle de réduction.

Définition 50. *Soit A et B deux langages binaires. On dit que A se réduit à B, au sens multivoque ($A \leq_m B$), si et seulement s'il existe une fonction $f : \Sigma_1^* \to \Sigma_2^*$ calculable telle que :*

$$\forall\, \omega \in \Sigma_1^*,\ \omega \in A \Leftrightarrow f(\omega) \in B.$$

Si la fonction f est calculable en espace logarithmique, on dit que A se réduit logarithmiquement à B au sens multivoque ($A \leq_m^{log} B$).

Définition 51. *Un langage L sur un alphabet Σ est* `NP-complet` *selon $\leq_m^{log}$ si :*

1. *L est élément de* NP,
2. $\forall\, A \in \mathrm{NP},\ A \leq_m^{log} L$.

Nous utiliserons `NP-complet` pour signifier `NP-complet` selon la réduction multivoque en espace logarithmique.

La complétude de plusieurs problèmes, pour la classe NP, a depuis été démontrée. Nous vous référons au livre de Garey et Johnson [GJ90] pour obtenir une liste non-exhaustive de tels problèmes. L'avantage de connaître plusieurs problèmes `NP-complet` est illustré dans la proposition suivante.

Proposition 52. *Si A est* `NP-complet`, *$B \in$ NP et $A \leq_m^{log} B$ alors B est* `NP-complet`.

Ainsi, plus on connaît de langages A qui sont NP-complet, plus il sera facile de trouver un bon candidat afin de montrer que B est NP-complet. Dans le cas particulier qui nous concerne, le problème suivant nous permettra d'obtenir le résultat recherché. Précisons préalablement qu'une formule 3-FNC est dite monotone si elle ne contient que des littéraux positifs.

Définition 53. *One-in-Three monotone 3SAT (*1in3m3SAT*) :*

Entrée : F, *une formule monotone 3-FNC.*

Question : Existe-t-il une affectation des variables telle que pour chaque clause de F, *il y a exactement une variable sur trois de satisfaite ?*

Schaefer [Sch78] a démontré que le problème 1in3m3SAT est NP-complet en utilisant son théorème de la dichotomie.

Proposition 54. $\mathtt{A}_{\mathbb{N}}$ *est* NP-complet *selon la réduction multivoque en espace logarithmique.*

Démonstration. Pour montrer que $\mathtt{1in3m3SAT} \leq_m^p \mathtt{A}_{\mathbb{N}}$, il suffit de construire une fonction calculable en espace logarithmique, f, telle que $W \in \mathtt{1in3m3SAT}$ si et seulement si $f(W) \in \mathtt{A}_{\mathbb{N}}$ pour toute entrée W.
Soit W une formule 3FNC monotone avec m variables, on peut écrire W sous la forme :

$$W = C_1 \wedge C_2 \wedge \ldots \wedge C_k,$$

où chaque clause de W est de la forme $C_i = w_{i,1} \vee w_{i,2} \vee w_{i,3}$.
Considérons maintenant la fonction f définie par $f(W) = \langle 1^n, \hat{M} = (M, \mu, \tau)\rangle$ où la matrice M possède k lignes et m colonnes. Chaque variable est affectée à une colonne selon l'ordre des indices, c'est-à-dire que $\mu(j) = x_j$. La valeur de $M_{s,t}$ est 1 si x_t est dans la clause C_s et 0 sinon. Le vecteur cible de $\hat{M}$ est 1^k.
Pour construire $\langle 1^n, \hat{M}\rangle$, nous avons donc besoin de trois compteurs bornés, en valeur, par $\mathtt{max}\{k, n\}$. L'espace utilisé est donc dans $O(lg\ \mathtt{max}\{k, n\}) = O(lg\ \mid W \mid)$.
Il reste à montrer que $W \in \mathtt{1in3m3SAT} \Leftrightarrow f(W) = \langle 1^n, \hat{M}\rangle \in \mathtt{A}_{\mathbb{N}}$ où n est le nombre de variables de W.
($\Rightarrow$) Si $W \in \mathtt{1in3m3SAT}$, alors il existe $x \in \{0,1\}^n$ tel que $x \models W$ et, pour toutes les clauses de W, exactement un littéral est satisfait. Il est clair, par la construction, que

la somme des colonnes de M^x donne exactement 1^k. Ainsi, puisque M^{1^n} contient les colonnes de M^x, la même combinaison fonctionne également et donc 1^k est généré par M^{1^n} et donc $\hat{M}$ accepte 1^n et $f(W) \in \mathtt{A}_{\mathbb{N}}$.

($\Leftarrow$) Si $f(W) \in \mathtt{A}_{\mathbb{N}}$, alors $\hat{M}$ accepte 1^n. Il existe donc une combinaison linéaire des colonnes de M^{1^n} qui est égale à 1^k. Cette combinaison peut s'écrire sous la forme $S = \sum_{i=1}^{n} \alpha_i M_i = 1^k$, où les M_i sont les colonnes de M^{1^n} et les $\alpha_i \in \{0,1\}$. Il est clair qu'utiliser un α_i plus grand que 1 est impossible car pour chaque position de M_i égale à 1 on aurait $S_i > 1$ ce qui contredit $S = 1^k$. De plus, puisque $S = 1^k$, il suit que pour chaque position $i \in [k]$, il existe exactement une colonne M_j telle que $\alpha_j = 1$ et $M_j(i) = 1$ car sinon on aurait $S_i = 0$ ou $S_i > 1$. On conclut que $x \in \{0,1\}^n$ défini par $x_j = \alpha_j$ est un témoin de $W \in \mathtt{1in3m3SAT}$.

On termine la preuve en appliquant la proposition 52 avec $A = \mathtt{1in3m3SAT}$ et $B = \mathtt{A}_{\mathbb{N}}$.

■

2.3.3 Le semi-anneau des graphes

Le semi-anneau étudié dans cette section nous est moins familier. Considérons un ensemble V de v éléments appelés sommets. On peut alors identifier V avec l'ensemble $[v] = \{1, 2, \ldots, v\}$. On définit le semi-anneau des graphes orientés sur V par $\mathbb{G}(v) = (\wp([v]^2), \cup, \odot)$ où les opérations sont définies comme suit :

Définition 55. *Soit $\alpha, \beta \in \wp([v]^2)$, alors :*

- $\alpha \cup \beta = \{(u,w) \mid (u,w) \in \alpha \text{ ou } (u,w) \in \beta\}$
- $\alpha \odot \beta = \{(u,w) \mid \exists c \in V \text{ tel que } (u,c) \in \alpha \text{ et } (c,w) \in \beta\}$

L'élément neutre pour l'addition est donc l'ensemble vide et l'élément neutre pour la multiplication est l'ensemble contenant exactement les éléments (u,u) pour $u \in [v]$. Nous allons démontrer que le problème $\mathtt{A}_{\mathbb{G}(v)}$ est toujours dans AC^0 sans égard au nombre de sommets.

On peut vérifier que l'opération $\odot$ est distributive à droite :

$$\forall\, \alpha, \beta, \gamma \in \mathbb{G}(v) \quad (\alpha \cup \beta) \odot \gamma = (\alpha \odot \gamma) \cup (\beta \odot \gamma).$$

On remarque également que les opérations $\cup$ et $\odot$ sont monotones et que

$$\forall\, \alpha, \beta, \gamma \in \mathbb{G}(v) \quad \alpha \odot \gamma \subseteq (\alpha \cup \beta) \odot \gamma.$$

Définition 56. *Soit $k \in \mathbb{N}$ et $\alpha, \beta \in \mathbb{G}(v)^k$, on dit que $\alpha \subseteq \beta$ si et seulement si $\alpha[i] \subseteq \beta[i]$ pour tout i dans $[k]$.*

On remarque que la relation $\subseteq$ définie sur $\mathbb{G}(v)^k$ est un ordre partiel. Ainsi, soit $\alpha \in G(v)^k$ tel que $\beta \subseteq \alpha$ pour tout graphe $\beta \in A \subseteq G(v)^k$ alors α est une borne supérieure de A. Nous ne sommes toutefois pas en présence d'un TBD puisque $G(v)$ n'est pas un treillis. En effet, $a = a \cup (a \otimes b)$ si et seulement si $a \supseteq a \otimes b$ ce qui n'est clairement pas tout le temps de cas.

Proposition 57. *Soit $k \in \mathbb{N}$, $\alpha, \beta \in \mathbb{G}(v)$ et $l, \tau \in \mathbb{G}(v)^k$, on a alors*

$$\alpha \otimes l \subseteq \tau \text{ et } \beta \otimes l \subseteq \tau \Leftrightarrow (\alpha \cup \beta) \otimes l \subseteq \tau.$$

Démonstration. Si $\alpha \otimes l \subseteq \tau$ et $\beta \otimes l \subseteq \tau$ alors $((\alpha \otimes l) \cup (\beta \otimes l)) \subseteq \tau$. En utilisant la distributivité à droite, on obtient $(\alpha \cup \beta) \otimes l \subseteq \tau$.

Supposons que $(\alpha \cup \beta) \otimes l \subseteq \tau$, on sait que $\alpha \otimes l, \beta \otimes l \subseteq (\alpha \otimes l) \cup (\beta \otimes l) = (\alpha \cup \beta) \otimes l$. En combinant ces inclusions on obtient $\alpha \otimes l \subseteq \tau$ et $\beta \otimes l \subseteq \tau$. ■

Dans le cadre des programmes de génération sur un graphe, on peut traduire la proposition précédente comme suit :

Soit $\hat{M}$ un programme de génération sur $\mathbb{G}(v)$. Soit M_i une colonne de M^x avec $x \in \{0,1\}^n$. Si τ est une combinaison linéaire des colonnes de M^x dont le coefficient de M_i est α et τ est une combinaison des colonnes de M^x dont le coefficient de M_i est β alors τ est également une combinaison linéaire des colonnes de M^x avec $\alpha \cup \beta$ comme coefficient de M_i. La proposition permet également de conclure que si $\alpha \otimes M_i \subseteq \tau$ alors $\alpha_i \otimes M_i \subseteq \tau$ pour tout sous ensemble α_i de α.

En combinant ces observations, nous avons conçu l'algorithme 2 qui permet d'évaluer $\mathtt{A}_{\mathbb{G}(v)}$ sur une entrée $\langle x, \hat{M} \rangle$.

On supposera que m est le nombre de colonnes de M et que g_i est le graphe élémentaire contenant seulement le i-ème arc selon l'ordre lexicogaphique :

$$(1,1), (1,2), \ldots (1,v), (2,1), \ldots, (v,v).$$

On peut considérer un graphe g comme un vecteur de $\{0,1\}^{v^2}$ dont la i-ème composante est 1 si et seulement si g contient le i-ème arc selon cet ordre.

La proposition suivante démontre la validité de l'algorithme 2.

Algorithme 2 Algorithme pour décider $\mathtt{A}_{\mathbb{G}(v)}$.

N : sur entrée $\langle x, \hat{M} \rangle$
Calculer M^x
pour $i \leftarrow 1$ à m **faire**
  pour $j \leftarrow 1$ à v^2 **faire**
    si $g_j \otimes M_i \subseteq \tau$ et M_i est une colonne de M^x **alors**
      $\alpha_i[j] = 1$ // le coefficient de la colonne i peut contenir le j-ème arc.
    sinon
      $\alpha_i[j] = 0$ // le coefficient de la colonne i ne peut contenir le j-ème arc
      // ou M_i n'est pas une colonne de M^x.
si $\bigcup_{i=0}^{m} \alpha_i \otimes M_i = \tau$ **alors**
  Retourner 1
sinon
  Retourner 0

Proposition 58. *Soit $n \in \mathbb{N}$, $\hat{M} = (M, \mu, \tau)$ un programme de génération sur $G(v)$ qui calcule la fonction $f_{\hat{M}} : \{0,1\}^n \to \{0,1\}$ et $x \in \{0,1\}^n$. Alors la valeur retournée par N sur une entrée $\langle x, \hat{M} \rangle$ est $f_{\hat{M}}(x) = \chi_{\mathtt{A}_{\mathbb{G}(v)}}(\langle x, \hat{M} \rangle)$.*

Démonstration. On remarque que chaque graphe g peut être décomposé en l'union de graphes élémentaires. En effet, il suffit de choisir exactement les graphes élémentaires correspondant aux arcs du graphe g.

Si M_i est une colonne de M^x, l'algorithme construit α_i comme l'union de tous les graphes élémentaires g_j tels que $g_j \otimes M_i \subseteq \tau$. On peut alors utiliser la proposition 57 et conclure que $\alpha_i \otimes M_i \subseteq \tau$. De plus, considérons un graphe β_i tel que $\beta_i \otimes M_i \subseteq \tau$. Alors tous les sous-graphes élémentaires, h_j, de β_i sont tels que $h_j \otimes M_i \subseteq \tau$. Ainsi, $\beta_i \subseteq \alpha_i$ car chaque sous-graphe élémentaire de β_i est également un sous-graphe élémentaire de α_i. Le graphe α_i est donc une borne supérieure parmi les graphes β_i tels que $\beta_i \otimes M_i \subseteq \tau$. Ainsi, pour toute famille, (β_i) de coefficients dans $\mathbb{G}(v)$, soit

$$\bigcup_{i=0}^{m} \beta_i \otimes M_i \subseteq \bigcup_{i=0}^{m} \alpha_i \otimes M_i$$

soit

$$\tau \subsetneq \bigcup_{i=0}^{m} \beta_i \otimes M_i.$$

On conclut donc que si $\bigcup_{i=0}^{m} \alpha_i \otimes M_i \neq \tau$ alors $f_{\hat{M}} = 0$. De plus, il est clair, par construction, que si $\bigcup_{i=0}^{m} \alpha_i \otimes M_i = \tau$ alors $f_{\hat{M}} = 1$, ce qui termine la preuve. ■

L'analyse de la complexité de l'algorithme 2 permet de démontrer la proposition suivante. Comme le raisonnement est similaire à la démonstration de la proposition 49, nous nous contentons de donner les grandes lignes du raisonnement.

Proposition 59. *Le problème* $\mathtt{A}_{\mathbb{G}(v)}$ *est dans* AC^0.

Démonstration. Considérons une entrée $\langle x, \hat{M} \rangle$ pour le problème $\mathtt{A}_{\mathbb{G}(v)}$. On a déjà vu que calculer les colonnes de M^x est possible en profondeur constante. De plus, comme le graphe $\mathbb{G}(v)$ est fixé, il devient possible d'effectuer le calcul des boucles des lignes 3 et 4 en $m \cdot v^2$ circuits parallèles. Pour i et j fixés, comme $\mathbb{G}(v)$ est fixé et fini, la ligne 5 se calcule par un circuit de profondeur constante. Le calcul des huit premières lignes de l'algorithme est donc possible par un circuit de profondeur constante et de taille polynomiale par rapport à $\langle x, \hat{M} \rangle$. À ce moment, le circuit possède $m \cdot v^2$ portes de sortie telles que la l-ème porte est vraie si et seulement si $\alpha_i[j] = 1$ où $i \cdot j = l$. C'est-à-dire qu'à ce moment, nous connaissons la valeur de tous les α_i. Pour calculer la ligne 9, on utilise m copies d'un circuit calculant $\alpha_i \otimes M_i$. Chacun de ces circuits possède v^2 portes de sortie qui correspondent aux v^2 composantes d'un élément de $\mathbb{G}(v)$. Il reste ensuite à faire l'union des résultats et de comparer le résultat avec τ.

On conclut qu'il est possible d'implémenter l'algorithme 2 par une famille de circuits de taille polynomiale et de profondeur constante et donc que le problème $\mathtt{A}_{\mathbb{G}(v)}$ est dans AC^0. ■

Chapitre 3

Machine de Turing algébrique

La complexité calculatoire est l'étude des ressources nécessaires lors du calcul d'une fonction par un modèle particulier. Dans le présent chapitre, le modèle de calcul étudié est la machine de Turing algébrique. Nous avons déjà mentionné que la machine de Turing algébrique a été introduite par Damm *et al.* [DHM02] à des fins utilitaires. Il semble donc qu'une étude dédiée à la machine de Turing algébrique soit pertinente. Loin d'épuiser le sujet, ce chapitre se veut plutôt une introduction à la machine de Turing algébrique et pourra servir de base à une étude plus exhaustive.

Ce chapitre sera divisé en trois sections. Les deux premières sections consistent essentiellement en une adaptation de théorèmes classiques sur la machine de Turing à la variante algébrique. Le manuel de Hopcroft et Ullman [HU79, Chapitre 12] sera utilisé à titre de référence. La réputation de ce manuel n'étant plus à faire, le choix de cette référence est tout désigné afin de cibler les théorèmes considérés comme classiques pour la machine de Turing. La dernière section consiste en une discussion sur la machine de Turing algébrique alternante.

La définition suivante sera utile dans ce chapitre.

Définition 60. *Une fonction $f : \mathbb{N} \to \mathbb{N}$ est dite « entièrement constructible en espace » s'il existe une machine de Turing déterministe M utilisant exactement $f(|x|)$ cellules de son ruban de calcul pour toute entrée $x \in \{0,1\}^*$.*

Notons que les fonctions habituellement étudiées possèdent cette propriété. En particulier, les polynômes forment une famille de fonctions entièrement constructibles en

espace. Il n'est donc pas abusif de supposer qu'une fonction est entièrement constructible en espace.

3.1 Réduction de ressources sans incidence

Dans le manuel de Hopcroft et Ullman, il est d'abord démontré que l'existence d'une machine de Turing utilisant certaines ressources pour calculer une certaine fonction implique l'existence d'une autre machine de Turing calculant la même fonction tout en utilisant moins de ressources. La présente section sera consacrée à l'adaptation de ces théorèmes pour la machine de Turing algébrique.

3.1.1 L'espace

Théorème 61 (Compression de ruban). *Soit S un semi-anneau. Si $f : \{0,1\}^* \to S$ est calculée par une machine de Turing algébrique utilisant au plus $s(n) > n$ cellules durant son calcul sur toute entrée $x \in \{0,1\}^n$, alors, pour toute constante $c > 0$, f est calculée par une machine de Turing algébrique qui utilise au plus $c \cdot s(n)$ cellules durant son calcul sur toute entrée $x \in \{0,1\}^n$.*

La preuve suivante est pour une machine de Turing avec un seul ruban pour la lecture et l'écriture, ce qui justifie de prendre $s(n) > n$. On peut facilement adapter cette preuve pour une machine à plusieurs rubans en utilisant plus d'états pour la machine M_2. Il est également possible d'étendre la preuve au cas $s(n) > lg(n)$, c'est-à-dire à une machine de Turing possédant un ruban pour l'entrée (en lecture seulement) et un ruban pour le calcul.

Démonstration. Considérons une S-MTA $M_1 = (Q_1, \{0,1\}, \Gamma_1, \delta_1, q_0, B, q_f)$ qui calcule f en utilisant au plus $s(n)$ cellules durant le calcul pour une entrée $x \in \{0,1\}^n$. Nous allons construire une S-MTA $M_2 = (Q_2, \{0,1\}, \Gamma_2, \delta_2, q_0', B, q_f')$ qui simule M_1 et telle que, pour un certain entier r, une cellule du ruban de M_2 contient un symbole représentant r cellules consécutives du ruban de M_1.

On pose $\Gamma_2 = \Gamma_1^r$. Pour un mot $x \in \{0,1\}^n \subseteq \Gamma_1^n$, on note $\rho(x)$ l'encodage de x sur $\Gamma_2^{\lceil n/r \rceil}$. Le nombre r étant fixé, M_2 peut calculer $\rho(x)$ en regardant son entrée et en utilisant son contrôle fini. Ainsi, M_2 n'utilise que les n cellules de l'entrée afin de

l'encoder sur $\lceil n/r \rceil$ cellules. Il faut identifier B^k avec B afin de ne pas avoir à encoder tout le ruban infini.

On note Q_2^1 l'ensemble des états de M_2 nécessaire afin d'encoder l'entrée et de retourner au début du ruban. Sans expliciter les éléments de cet ensemble, mentionnons tout de même que le poids associé à chaque transition sortant d'un état de Q_2^1 est 1. Cette partie du calcul est déterministe ce qui permet de conclure que le poids total associé à l'encodage de l'entrée est 1.

Pendant la simulation de M_1, la machine M_2 utilise son contrôle fini afin de déterminer laquelle des r cellules de M_1 est réellement pointée par sa tête de lecture. Pour ce faire, M_2 a besoin de l'ensemble d'états $Q_2^2 = Q_1 \times [r]$ où l'état (q_i, j) signifie que la machine M_1 est dans l'état q_i et pointe sur le j-ème caractère parmi les r encodés par la cellule lue par M_2.

On conclut que l'ensemble des états de M_2 est $Q_2 = Q_2^1 \cup Q_2^2 \cup \{q'_f\}$.

La transition marquant la fin de l'encodage de l'entrée et le début de la simulation de M_1 est de la forme

$$(q, \alpha, (q_0, 1), \alpha, S, 1),$$

où q est l'état dans lequel on se retrouve à la fin de l'encodage de l'entrée et α est le caractère sur la première cellule de M_2. Lorsque M_2 entre dans un état de la forme (q_f, j), la simulation de M_1 est terminée et M_2 peut alors arrêter son calcul. Ainsi, à la fin de la simulation de M_1, M_2 effectue une étape de calcul de plus qui est de la forme

$$((q_f, j), \alpha, q'_f, \alpha, H, 1),$$

pour $j \in [r]$ et $\alpha \in \Gamma_2$.

La relation de transition de M_2 pour les états de Q_2^2 peut maintenant être définie. Considérons les déplacements de la tête de M_2 qui sont possibles. Pour $i, j \in [r]$, on définit

$$D(i,j) = \begin{cases} L & \text{si } i = 1 \text{ et } j = r \\ H & \text{si } 1 < i, j < r. \\ D & \text{si } i = r \text{ et } j = 1. \end{cases}$$

Puisque la tête de M_1 ne peut se déplacer que d'une cellule au plus à la fois, les autres cas pour i et j sont impossibles.

Soit $(q_1, i), (q_2, j) \in Q_2$, $\alpha, \beta \in \Gamma_2$ et $s \in S$, alors

$$((q_1, i), \alpha, (q_2, j), \beta, D(i, j), s) \in \delta_2$$

si et seulement si

$$(q_1, \rho^{-1}(\alpha)[i], q_2, \rho^{-1}(\beta)[j], D'(i, j), s) \in \delta_1,$$

où

$$D'(i,j) = \begin{cases} L & \text{si } D(i,j) = L \text{ ou } D(i,j) = H \text{ et } i > j \\ D & \text{si } D(i,j) = D \text{ ou } D(i,j) = H \text{ et } i < j \\ H & \text{si } i = j. \end{cases}$$

Notons que $\rho^{-1}(\alpha)[i]$ signifie : « le i-ème caractère de la préimage de α par ρ ». Il y a donc une correspondance, qui préserve le poids $s \in S$, entre les éléments de δ_2 qui ne contiennent que des états de Q_2^2 et les éléments de δ_1.

Ainsi, pour une entrée $x \in \{0,1\}^*$, le poids retourné par M_2 sera le même que le poids retourné par M_1. En effet, l'initialisation de M_2 est déterministe et a poids 1. À partir de la configuration

$$(\epsilon, (q_0, 1), \rho(x))$$

le calcul de M_2 imite le calcul de M_1 étape par étape en utilisant les mêmes poids. À la fin de la simulation, on ajoute une étape de calcul de poids 1. La fonction calculée par M_2 est donc :

$$f_{M_2}^S(x) = 1 \cdot f_{M_1}^S(x) \cdot 1 = f_{M_1}^S(x) = f(x).$$

L'espace utilisé par M_2 dépend du choix de r. En choisissant r tel que $r \cdot c > 2$, on obtient une machine de Turing algébrique M_2 qui utilise au plus $\lceil \frac{s(n)}{r} \rceil \leq \frac{c \cdot s(n)}{2}$ cellules si $s(n) > r$ et une seule cellule sinon. ■

Le théorème suivant affirme que le nombre de rubans utilisés par une machine de Turing algébrique n'affecte pas l'espace nécessaire afin de calculer une fonction.

Théorème 62 (Réduction du nombre de rubans). *Soit S un semi-anneau. Si $f : \{0,1\}^* \to S$ est calculée par une S-MTA à k rubans utilisant au plus $s(n) > n$ cellules alors f est calculé par une S-MTA à un ruban (de calcul) utilisant un espace au plus $s(n)$ cellules.*

Démonstration. Soit $M_1 = (Q_1, \{0,1\}, \Gamma_1, \delta_1, q_0, B, q_f)$ une S-MTA à k rubans qui calcule f en utilisant au plus $s(n)$ cellules durant le calcul pour une entrée $x \in \{0,1\}^n$. Nous allons construire une S-MTA $M_2 = (Q_2, \{0,1\}, \Gamma_2, \delta_2, q_0', B, q_f')$ avec un seul ruban qui simule M_1 en utilisant $2 \cdot s(n)$ cellules. Il suffit ensuite d'utiliser le théorème 61, avec $c = 1/2$, pour conclure qu'il existe une S-MTA à un ruban qui calcule f en utilisant au plus $s(n)$ cellules.

On choisit $\Gamma_2 = \Gamma_1^k \times \{0,1\}^k$. M_2 utilisera un symbole de Γ_2 sur une cellule i afin d'encoder les symboles que M_1 contient sur la cellule i de ses k rubans et les têtes qui pointent sur la i-ème cellule de leur ruban. Encore une fois, on identifie le symbole $(B^k, 0^k)$ avec le symbole B afin de ne pas avoir à encoder les cellules qui ne sont pas utilisées par M_1.

L'initialisation de M_2 s'effectue comme suit. M_2 regarde sa première cellule et modifie le caractère lu, $\alpha \in \{0,1\}$, par le caractère $(\alpha, B, B, \ldots, B), 0^k) \in \Gamma_2$. Ici, le 1^k signifie que les k têtes de M_1 pointent sur la première case de leur ruban respectif. M_2 parcourt ensuite son entrée en laissant une cellule, dite temporaire, entre chaque deux cellules utilisées et transforme chaque symbole $\alpha \in \{0,1\}$ de son entrée par un symbole $((\alpha, B, B, \ldots, B), 0^k) \in \Gamma_2$. M_2 retourne ensuite au début de son ruban. L'initialisation est déterministe et M_2 associe le poids 1 à chaque transition ce qui donne un poids total de 1.

Les cellules temporaires seront utilisées pendant la simulation d'une étape de M_1. Avant la simulation d'une étape de M_1, les cellules temporaires sont toujours vides. Afin d'éviter la confusion, le terme « cellule active » sera utilisé pour parler d'une cellule qui n'est pas temporaire. On associe une cellule temporaire à la cellule active directement à sa droite.

La simulation d'une étape du calcul de M_1 est décrite dans l'algorithme 3. Il est à noter que cette simulation peut entièrement être inscrite dans le contrôle fini de M_2 en remplaçant un état $q \in Q_1$ par un ensemble d'états $Q(q) \subset Q_2$ qui simulera une étape de calcul de M_1 à partir de q. Les détails de cette simulation sont donnés dans l'algorithme 3.

Toutes les étapes de l'algorithme 3 ont poids 1 sauf la 5-ème qui a poids ω, où ω est le poids de la transition simulée. Ainsi, le poids associé à la simulation d'une transition de

Algorithme 3 La simulation d'une étape de calcul de M_1 à partir de q.

1: Lire le ruban afin de connaître la position des k têtes,
2: Revenir au début du ruban
3: // On a alors identifié la transition T à simuler et son poids ω.
4: Copier le contenu de la première cellule active dans sa cellule temporaire
5: Effacer les têtes de la première cellule temporaire
6: **pour** chaque cellule active c_i du ruban de M_2 **faire**
7: Modifier la cellule temporaire de la cellule active de gauche suivant T
8: // i.e. ajouter les têtes qui se déplacent vers la gauche.
9: Modifier la cellule temporaire de c_i suivant T
10: // i.e. modifier le contenu des cellules i des k rubans de M_1 et ajouter
11: // les têtes qui restent sur place.
12: Modifier la cellule temporaire de la cellule active à droite suivant T
13: // i.e. ajouter les têtes qui se déplacent vers la droite.
14: Mettre à jour les cellules actives suivant leur cellule temporaire
15: Effacer le contenu des cellules temporaires, revenir au début du ruban

M_1 est le même que le poids de la transition simulée. Comme dans la preuve précédente, M_2 termine sont calcul par une étape de poids 1 pour entrer dans son état final.

La machine M_2 utilise donc au plus $2 \cdot s(n)$ cellules et la fonction calculée par M_2 est bien

$$f^S_{M_2}(x) = 1 \cdot f^S_{M_1}(x) \cdot 1 = f^S_{M_1}(x) = f(x),$$

ce qui conclut la preuve. ■

3.1.2 Le temps

Théorème 63. *Si $f : \{0,1\}^* \to S$ est calculée par une machine de Turing algébrique à k rubans et utilisant au plus $t(n)$ étapes de calcul, où $inf_{n\to\infty} \frac{t(n)}{n} = 0$ et $k > 2$, alors f est calculée par une machine de Turing algébrique à k rubans utilisant au plus $c \cdot t(n)$ étapes de calcul pour toute constante $c > 0$.*

Nous nous contentons de décrire brièvement la preuve et de démontrer sa validité. Nous vous référons au manuel de Hopcroft et Ullman [HU79] pour les détails techniques

qui décrivent le temps de calcul de la machine construite.

Démonstration. Considérons une S-MTA $M_1 = (Q_1, \{0,1\}, \Gamma_1, \delta_1, q_0, B, q_f)$ qui calcule f en utilisant au plus $t(n)$ étapes de calcul durant le calcul pour une entrée $x \in \{0,1\}^n$. Nous allons construire une S-MTA $M_2 = (Q_2, \{0,1\}, \Gamma_2, \delta_2, q_0', B, q_f')$ qui simule M_1 et telle que, pour un certain entier r, une cellule du ruban de M_2 contient un symbole représentant r cellules consécutives du ruban de M_1.

Comme dans la preuve du théorème 61, l'alphabet de ruban de M_2 est $\Gamma_2 = \Gamma_1^r$. Sur une entrée $x \in \{0,1\}^n$, M_2 calcule $\rho(x)$ et écrit le résultat sur le second ruban tout en effaçant le premier ruban. M_2 transfère ensuite le contenu du second ruban sur le premier ruban. M_2 a besoin de $2 \cdot (n + \lceil n/r \rceil)$ étapes de calcul pour initialiser son calcul.

Une machine de Turing algébrique à un ruban prendrait $O(n^2)$ étapes de calcul pour effectuer cet encodage. Cela justifie que le théorème ne tienne que pour $k > 2$.

On note Q_2^1 l'ensemble des états de M_2 nécessaires afin d'encoder l'entrée et de retourner au début du ruban. Sans expliciter les éléments de cet ensemble, mentionnons tout de même que le poids associé à chaque transition sortant d'un état de Q_2^1 est 1. Cette partie du calcul est déterministe ce qui permet de conclure que le poids total associé à l'encodage de l'entrée est 1.

Tout comme la machine de la preuve du théorème 61, M_2 utilise son contrôle fini afin de savoir lequel des r symboles encodés est pointé par la tête de M_1.

M_2 simule r transitions consécutives du calcul de M_1 en sept transitions. Ces transitions vont comme suit pour chaque ruban :

Algorithme 4 La simulation de r étapes de calcul de M_1 par M_2.

1: Lire la cellule et déplacer la tête vers la gauche
2: Lire la cellule et déplacer la tête vers la droite
3: Déplacer la tête vers la droite
4: Lire la cellule, modifier la cellule et déplacer la tête à gauche
5: Modifier la cellule et déplacer la tête vers la gauche
6: Modifier la cellule et déplacer la tête vers la droite
7: Bouger la tête en fonction de la position de la tête de M_1

Après les quatre premières étapes, M_2 connaît le contenu des cellules pouvant être

affectées par la simulation à venir. M_2 modifie alors les trois cellules en fonction des r transitions de M_1 à simuler. Le calcul des modifications à apporter est inscrit dans le contrôle fini de M_2 et ne prend donc pas de temps. La dernière transition dépend du déplacement global de la tête de M_1 après les r étapes simulées.

On associe le poids 1 aux transitions un à six. Le poids de la septième transition est le produit du poids des r transitions de M_1 que M_2 simule.

À la fin de la simulation, M_2 entre dans son état final avec poids 1. La fonction calculée par M_2 est donc

$$f^S_{M_2}(x) = 1 \cdot f^S_{M_1}(x) \cdot 1 = f^S_{M_1}(x) = f(x),$$

ce qui conclut la preuve de la validité. ■

3.2 Plus de ressources augmente la puissance

Dans cette section, nous construirons une hiérarchie pour la machine de Turing algébrique. La hiérarchie consiste simplement en la démonstration qu'il est possible d'augmenter la puissance expressive d'une machine de Turing algébrique en augmentant suffisamment ses ressources.

Inspirée de la hiérarchie non-déterministe, la construction s'effectuera en deux étapes : l'adaptation du théorème de Savitch et la démonstration de l'existence de la hiérarchie spatiale pour la machine de Turing algébrique.

3.2.1 Le théorème de Savitch

Le théorème de Savitch constitue certainement un résultat fondamental de la théorie de la complexité du calcul. Essentiellement, le théorème affirme qu'il est toujours possible de simuler une `MTND` utilisant un espace dans $O(s(n))$ par une machine de Turing déterministe (`MTD`) utilisant un espace dans $O(s^2(n))$.

Ce théorème a ceci d'étonnant que la meilleure borne supérieure pour simuler une `MTND` utilisant un temps $O(t(n))$ par une `MTD` est $O(2^{O(t(n))})$.

Ainsi, alors que seule l'inclusion «NP $\subseteq$ EXPTIME» est connue pour le temps de calcul, le théorème de Savitch permet de conclure, en particulier, que «PSPACE =

NPSPACE». Rappelons que la classe PSPACE est la classe des problèmes décidables en espace déterministe polynomial et que NPSPACE est son équivalent non-déterministe.

Le coeur de la preuve du théorème de Savitch est un algorithme qui détermine si un graphe orienté possède un chemin entre deux sommets donnés. Voici une généralisation de l'algorithme de Savitch adaptée à la machine de Turing algébrique. L'algorithme $M^n_{Savitch}$ est conçu pour une S-MTA M, et une longueur d'entrée n fixée. La relation de transition de M ainsi que l'ensemble des configurations de M possibles pour une entrée de longueur n sont inscrits dans le contrôle fini de $M^n_{Savitch}$. L'entrée de l'algorithme est $\langle C_a, C_b, d, x\rangle$ où C_a et C_b sont deux configurations de la machine simulée, d est un entier positif et $x \in \{0,1\}^n$. La valeur retournée par l'algorithme est la somme du poids des chemins de longueur exactement 2^d entre C_a et C_b qui sont consistants avec x.

Algorithme 5 Algorithme de Savitch généralisé.

1: $M^n_{Savitch}$: sur entrée $\langle C_a, C_b, d, x\rangle$
2: $poids \leftarrow 0$
3: **si** $d = 0$ et $C_a \vdash^x C_b$ **alors**
4: **retourner** $\omega(C_a, C_b, x)$
5: **pour** toute configuration C_i de G_n **faire**
6: $poids \leftarrow poids + [M^n_{Savitch}(\langle C_a, C_i, d-1, x\rangle) \cdot M^n_{Savitch}(\langle C_i, C_b, d-1, x\rangle)]$
7: **retourner** *poids*

Pour M fixé, on remarque que la description de $M^n_{Savitch}$ est calculable en espace logarithmique par rapport à n. En effet, la seule partie de $M^n_{Savitch}$ affectée par la valeur de n est l'ensemble des configurations G_n et G_n est calculable en espace logarithmique par rapport à n.

Proposition 64. *Soit M, une S-MTA, C_a et C_b deux configurations de M. Alors l'algorithme $M^n_{Savitch}$ appliqué à l'entrée $\langle C_a, C_b, d, x\rangle$ retourne la somme du poids des chemins, consistants avec x, de la configuration C_a à la configuration C_b de longueur exactement 2^d.*

Nous utiliserons $\Phi^x_l(C_a, C_b)$ pour désigner l'ensemble des chemins, consistants avec

x, de C_a à C_b qui ont exactement une longueur l. Par $W(\Phi_l^x(C_a, C_b))$, nous désignons la somme du poids des chemins dans $\Phi_l^x(C_a, C_b)$. De plus, nous utiliserons la notation $\Phi_l^x(C_a, C, C_b)$ pour désigner l'ensemble des chemins, consistants avec x, de longueur l de C_a à C_b tels que la $(l/2)$-ème configuration visitée est C.

Démonstration. **Base :** Soit $d = 0$, alors $M^n_{Savitch}(\langle C_a, C_b, d, x\rangle) = \omega(C_a, C_b, x)$ et

$$\Phi^x_{2^d}(C_a, C_b) = \Phi^x_1(C_a, C_b)$$

est l'ensemble des transitions de C_a à C_b qui sont consistantes avec x ainsi

$$W(\Phi^x_{2^d}(C_a, C_b)) = \omega(C_a, C_b, x).$$

Pas d'induction : Soit $d > 0$, supposons que $M^n_{Savitch}(\langle C_1, C_2, d-1, x\rangle) = W(\Phi^x_{2^{d-1}}(C_1, C_2))$ pour toutes configurations C_1 et C_2. On peut diviser les chemins de C_a à C_b de longueur 2^d en deux parties de longueur 2^{d-1}. On sait également que le poids d'un chemin est le produit des arcs qui le composent et que le produit est associatif dans un semi-anneau. On obtient ainsi :

$$\begin{aligned} W(\Phi^x_{2^d}(C_a, C_b)) &= \sum_{config\ C} \left(\sum_{\phi \in \Phi^x_{2^d}(C_a, C, C_b)} W(\phi) \right) \\ &= \sum_{config\ C} \left(\sum_{\phi \in \Phi^x_{2^{d-1}}(C_a, C)} W(\phi) \cdot \sum_{\phi \in \Phi^x_{2^{d-1}}(C, C_b)} W(\phi) \right) \\ &= \sum_{config\ C} W(\Phi^x_{2^{d-1}}(C_a, C)) \cdot W(\Phi^x_{2^{d-1}}(C, C_b)) \end{aligned}$$

On peut alors appliquer l'hypothèse d'induction afin d'obtenir :

$$W(\Phi^x_{2^d}(C_a, C_b)) = \sum_{config\ C} M^n_{Savitch}(\langle C_a, C, d-1, x\rangle) \cdot M^n_{Savitch}(\langle C, C_b, d-1, x\rangle).$$

On observe que cette somme sur les configurations C est exactement le calcul effectué par l'algorithme $M^n_{Savitch}$ pour une entrée $\langle C_a, C_b, d, x\rangle$. On conclut que

$$W(\Phi^x_{2^d}(C_a, C_b)) = M^n_{Savitch}(\langle C_a, C_b, d, x\rangle).$$

■

Soit $M'_{Savitch}$ l'algorithme obtenu en ajoutant le test donné par l'algorithme 6 après la seconde ligne de $M^n_{Savitch}$. Cela a pour effet d'ajouter une boucle implicite sur la configuration initiale de M. Cette boucle permet d'ajouter la contribution des chemins de longueur inférieure à 2^d. Ainsi, sur entrée $\langle C_0, C_f, f, x\rangle$, l'algorithme modifié permet de calculer les chemins, consistants avec x, de longueur au plus 2^d entre C_0 et C_f et pour d assez grand, la valeur retournée est donc $f(x)$.

Algorithme 6 Test qui simule une boucle sur C_0

si $C_a = C_b = C_0$ **alors**
  retourner 1
fin si

Lemme 65. *Soit M, une S-MTA qui calcule $f : \{0,1\}^* \to \{0,1\}$ en utilisant au plus $s(n)$ cellules pour une entrée $x \in \{0,1\}^n$. Alors la valeur retournée par l'algorithme $M'_{Savitch}$ sur l'entrée $\langle C_0, C_f, s(|x|), x\rangle$ est $f(x)$.*

Nous avons donc un algorithme déterministe qui simule une machine de Turing algébrique.

Nous devons maintenant analyser la complexité de cet algorithme. En effet, l'intérêt de l'algorithme de Savitch est qu'il permet une simulation déterministe, efficace en espace, du calcul non-déterministe. Alors que la validité de l'algorithme ne dépend pas de S, il en est autrement pour sa complexité.

Ainsi, le nombre d'appels récursifs et le nombre d'éléments de S à mémoriser est exactement le même pour tous les semi-anneaux (i.e. ne dépendent que de la machine de Turing algébrique). Toutefois la taille de l'encodage des éléments à mémoriser ainsi que la complexité des opérations algébriques diffèrent selon le choix du semi-anneau S.

Nous avons choisi de limiter notre analyse aux semi-anneaux finis. Deux raisons ont justifié ce choix. L'analyse de la complexité de l'algorithme 5 pour les semi-anneaux finis est intéressante. Elle constitue en plus une base nécessaire à l'analyse de la complexité pour les semi-anneaux infinis.

Théorème 66. *Soit $s(n)$, une fonction entièrement constructible en espace, S, un semi-anneau fini et M, une S-MTA qui calcule $f : \{0,1\}^* \to S$ en espace $O(s(n))$ alors il existe une machine de Turing déterministe qui calcule f en utilisant un espace $O(s^2(n))$.*

Démonstration. Considérons une machine de Turing déterministe M_1 qui implémente l'algorithme 5. La machine aura besoin de 2 rubans : le premier pour la pile d'exécution, le second pour la pile du calcul de la variable *poids*.

À chaque appel récursif, il faut mémoriser les valeurs $\langle C_a, C_b, l\rangle$. Soit $x \in \{0,1\}^*$ fixé, et $n = |x|$, le nombre de configurations possibles pour M sur une entrée de longueur n est $2^{d\cdot(s(n))}$ où $d \in \mathbb{N}$. On ne mémorise pas le x car il est sur le ruban d'entrée et ne change pas durant la simulation.

Ainsi, il suffit de $O(s(n))$ cellules pour encoder le détail des configurations. Pour que la valeur retournée par M_1 sur l'entrée $\langle C_0(x), C_f, l, x\rangle$ soit $f(x)$, il suffit de poser $l = d\cdot(s(n))$. Notons que $s(n)$ étant entièrement constructible en espace, il est possible de calculer l en espace $O(s(n))$. De plus, l'encodage de l prend $O(s(n))$ cellules et le nombre d'appels récursifs est au plus $d\cdot(s(n))$. On conclut que l'espace utilisé sur le premier ruban est au plus $O(s^2(n))$.

Puisque S est fini, l'encodage d'un élément de S utilise un nombre constant de cellules. De plus, on peut inscrire les tables d'additions et de multiplications de S dans la machine M_1 ce qui permet de calculer les opérations de S en espace constant. À chaque appel récursif, on retient au plus trois éléments de S, ce qui permet de conclure que l'espace utilisé sur le second est $O(s(n))$. ■

3.2.2 Une hiérarchie algébrique conditionnelle

Il est maintenant possible de démontrer qu'une machine de Turing algébrique qui utilise sensiblement plus d'espace peut calculer plus de fonctions. Inspiré de la hiérarchie non-déterministe pour l'espace, notre hiérarchie repose sur notre généralisation du théorème de Savitch. Cela justifie l'utilisation du qualificatif « conditionnelle ».

Afin de démontrer le résultat principal de cette section, nous aurons besoin du lemme de translation. La preuve donnée dans [HU79] s'adapte directement et nous nous contentons donc d'énoncer ce lemme.

Lemme 67 (Translation). *Soit $s_1(n), s_2(n)$ et $f(n)$ des fonctions entièrement constructibles en espace telles que $s_2(n), f(n) \geq n$ alors*

$$S\text{-}\#SPACE(s_1(n)) \subseteq S\text{-}\#SPACE(s_2(n))$$

implique

$$S\text{-}\#SPACE(s_1(f(n))) \subseteq S\text{-}\#SPACE(s_2(f(n))).$$

Voici maintenant le théorème de la hiérarchie algébrique.

Théorème 68. *Soit $\delta > 0$ et $r \geq 0$ alors*

$$S\text{-}\#SPACE(n^r) \subsetneq S\text{-}\#SPACE(n^{r+\delta}).$$

Bien que notre preuve soit calquée sur celle de Hopcroft et Ullman [HU79], les étapes de la preuve trouvent leur justification dans l'adaptation des théorèmes présentés dans ce chapitre. Nous avons donc choisi de présenter ladite preuve dans ce mémoire.

Démonstration ([HU79], Chapitre 12). Soit $r \geq 0$, alors il existe $s, t \in \mathbb{N}$ tels que $r \leq \frac{s}{t}$ et $r + \delta \geq \frac{s+1}{t}$. Il suffit donc de montrer que

$$\forall_{s,t\in\mathbb{N}} \;\; S\text{-}\#SPACE(n^{s/t}) \subsetneq S\text{-}\#SPACE(n^{(s+1)/t}).$$

Supposons le contraire, c'est-à-dire que

$$S\text{-}\#SPACE(n^{(s+1)/t}) \subseteq S\text{-}\#SPACE(n^{s/t}),$$

on peut alors appliquer le lemme 67 avec $f(n) = n^{(s+i)\cdot t}$, pour $i \in [s]$, ce qui donne

$$S\text{-}\#SPACE(n^{(s+1)(s+i)}) \subseteq S\text{-}\#SPACE(n^{s(s+i)}). \tag{3.1}$$

Puisque $s(s+i) \leq (s+1)(s+i-1)$ pour tout $i \geq 1$, on obtient que

$$S\text{-}\#SPACE(n^{s(s+i)}) \subseteq S\text{-}\#SPACE(n^{(s+1)(s+i-1)}). \tag{3.2}$$

En combinant les équations 3.1 et 3.2, on a

$$\begin{aligned}
S\text{-}\#SPACE(n^{(s+1)(2\cdot s)}) &\subseteq S\text{-}\#SPACE(n^{s(2\cdot s)}) \\
&\subseteq S\text{-}\#SPACE(n^{(s+1)(2\cdot s-1)}) \\
&\subseteq S\text{-}\#SPACE(n^{s(2\cdot s-1)}) \\
&\subseteq \cdots \\
&\subseteq S\text{-}\#SPACE(n^{(s+1)\cdot s}) \\
&\subseteq S\text{-}\#SPACE(n^{s^2}).
\end{aligned}$$

Ceci est équivalent à

$$S\text{-}\#SPACE(n^{2s^2+2s}) \subseteq S\text{-}\#SPACE(n^{s^2}).$$

On sait également, par le théorème 66, que

$$S\text{-}\#SPACE(n^{s^2}) \subseteq DSPACE(n^{2s^2}),$$

et le théorème de la hiérarchie spatiale pour la machine de Turing déterministe [HU79, Chapitre 12, Théorème 12.8] nous donne que

$$DSPACE(n^{2s^2}) \subsetneq DSPACE(n^{2s^2+2s}).$$

Cela reste vrai si on se restreint à la sous-classe $S\text{-}DSPACE(n^{2s^2+2s})$ qui ne contient que les fonctions $f \in DSPACE(n^{2s^2+2s})$ à valeur dans S. Sachant que

$$S\text{-}DSPACE(n^{2s^2+2s}) \subseteq S\text{-}\#SPACE(n^{2s^2+2s}),$$

on obtient la contradiction suivante

$$S\text{-}\#SPACE(n^{2s^2+2s}) \subsetneq S\text{-}\#SPACE(n^{2s^2+2s}),$$

qui nous permet de conclure que

$$S\text{-}\#SPACE(n^{(s+1)/t}) \subsetneq S\text{-}\#SPACE(n^{s/t})$$

pour tout $s, t \in \mathbb{N}$. ■

3.3 La machine de Turing algébrique alternante

Dans cette section, nous verrons que toute la théorie de la machine de Turing non-déterministe n'est pas facilement adaptable à sa variante algébrique. C'est en particulier le cas pour la machine alternante.

Nous avons choisi de parler de la machine de Turing algébrique alternante car nous croyons qu'il s'agit d'une belle ouverture pour le futur.

L'alternation, une généralisation du non-déterminisme

Chandra *et al.* ont introduit le concept d'alternation dans [CKS81]. Dans cet article, ils ont voulu étendre le concept de non-déterminisme. L'idée consiste à associer un quantificateur, existentiel ou universel, à chaque état de la machine. Pour chaque entrée x, on considère l'arbre de calcul, $T_M(x)$, de la machine de Turing alternante M où la racine de $T_M(x)$ est $C_0(x)$. L'acceptation de x par M est déterminée comme suit à partir de la racine de $T_M(x)$:

1. Si la configuration est C_f (une feuille de $T_M(x)$), accepter ;
2. Si l'état de la configuration est un état existentiel, accepter si et seulement s'il existe un descendant de la configuration qui accepte ;
3. Si l'état de la configuration est un état universel, accepter si et seulement si tous les descendants de la configuration acceptent.

Selon cette définition, la machine de Turing non-déterministe peut être considérée comme une machine de Turing alternante dont tous les états sont existentiels.

Machine alternante algébrique ou machine algébrique alternante ?

Dans [BH01], Beaudry et Holzer ont défini une machine de Turing alternante algébrique (S-MTAA). Cette définition a été choisie pour les besoins de leur article et il n'est pas évident d'y voir une généralisation de la machine de Turing algébrique. Ils ont peut-être volontairement placé le qualificatif « alternante » avant « algébrique » car leur définition correspond à l'algébrisation de la machine de Turing alternante.

Définition 69. *Une machine de Turing alternante algébrique sur S est une machine de Turing alternante à laquelle on associe un poids, pris dans S, aux transitions qui entrent dans l'état final.*

Soit M une machine de Turing alternante algébrique et $x \in \{0,1\}^*$. Soit $T_M(x)$ l'arbre du calcul de M sur x, alors M associe le poids suivant, noté $W(T_M(x))$, à $T(x)$.

1. Si la configuration est C_f, retourner le poids de la transition entrante ;
2. Si l'état de la configuration est un état existentiel, retourner la somme du poids des configurations descendantes ;

3. Si l'état de la configuration est un état universel, retourner le produit du poids des configurations descendantes.

La fonction $f_M^S : \{0,1\}^* \to S$ calculée par M est :

$$f_M^S(x) = W(T_M(x)).$$

On remarque immédiatement que le calcul de la fonction $f_M^S : \{0,1\}^* \to S$ n'est pas effectué de la même façon que pour la machine de Turing algébrique. De plus, il n'est pas nécessairement vrai que la machine de Turing existentielle algébrique de [BH01] calcule exactement les mêmes fonctions que la machine de Turing algébrique. Cela est le cas pour les $\mathbb{Z}$-modules. En effet, on remarque qu'une machine de Turing algébrique M_1 sur un $\mathbb{Z}$-module peut toujours être simulée par une machine de Turing algébrique M_2 dont tous les poids sont 1, sauf pour la transition finale, en utilisant $|c|$ copies d'une transition de M_1 de poids c. Le signe du poids de la transition finale dépend du signe du poids du chemin simulé et est calculé comme dans la preuve de 18. Sa valeur absolue est la même que celle du poids de la transition correspondante dans M_1.

Une définition correspondant mieux à la machine de Turing algébrique serait souhaitable afin de profiter de la richesse des travaux reliant la machine de Turing alternante aux classes de complexité classique.

Voici la définition que nous proposons pour la machine de Turing algébrique alternante.

Définition 70. *Une machine de Turing algébrique alternante sur S est une S-MTA dont les états sont divisés en trois groupes : les états existentiels, les états universels et l'état acceptant (q_f). On définit la fonction $f_M^S : \{0,1\}^* \to S$ qui associe à x le poids de $C_0(x)$ dans l'arbre de calcul $T_M(x)$. Ce poids est calculé récursivement comme suit :*

1. *Si la configuration est C_f, associer le poids $1 \in S$;*
2. *Si l'état de la configuration C est un état existentiel, le poids associé à la configuration est*

$$\sum_{C_i} \omega(C, C_i) \cdot W(C_i),$$

où les C_i sont les descendants de C et $W(C_i)$ est le poids de C_i ;

3. *Si l'état de la configuration C est un état universel, le poids associé à la configuration est*

$$\sum_{C_i} \omega(C, C_i) \cdot W(C_i)$$

si tous les $W(C_i)$ sont non-nuls et 0 sinon.

Comme pour la machine de Turing algébrique, on définit $f_M : \{0,1\}^* \to \{0,1\}$, la fonction binaire calculée par la machine de Turing algébrique alternante M par :

$$f_M(x) = 0 \Leftrightarrow f_M^S(x) = 0.$$

Il devient possible de généraliser la caractérisation logique des machines de Turing alternantes. Pour cela, introduisons les quantificateurs algébriques suivants.

Définition 71. *Soit S un semi-anneau et Ψ une fonction à valeur dans S, alors on définit $\exists^S$ et $\forall^S$ comme suit :*

1. $(\exists^S x)\ \Psi(x)$ *est vrai si et seulement si* $\sum_x \Psi(x) \neq 0$,
2. $(\forall^S x)\ \Psi(x)$ *est vrai si et seulement si* $\forall x\ \Psi(x) \neq 0$ *et* $\sum_x \Psi(x) \neq 0$.

On dit qu'une S-MTAA alterne k fois (k-S-MTAA) si et seulement si, pour chaque chemin de $T(x)$, il y a au plus k alternations, c'est-à-dire au plus k configurations dont le prédécesseur n'est pas associé au même quantificateur.

On a défini la machine de Turing déterministe comme une machine dont tous les poids sont 1 et pour laquelle chaque configuration possède un unique successeur. Si on élimine la contrainte sur les poids, c'est-à-dire que l'on permet des poids $\omega \in S$, on obtient une machine de Turing déterministe qui dépend de S. On note $S\text{-}DTIME(t(n))$ la classe des fonctions booléennes qui sont calculées par une telle machine utilisant au plus $t(n)$ étapes pour une entrée de longueur n.

Sachant qu'un élément $a \in S$ est appelé diviseur de zéro s'il existe un élément $b \in S$ tel que $a \cdot b = 0$, on peut démontrer la proposition suivante.

Proposition 72. *Soit S un semi-anneau sans diviseur de zéro. Alors*

$$S\text{-}DTIME(t(n)) = DTIME(t(n)).$$

Démonstration. Si une machine de Turing M, déterministe sur S, rejette une entrée $x \in \{0,1\}^*$ alors il existe une transition de poids 0 qui est utilisée par M pendant son calcul. Le poids exact d'une transition de poids non nul n'importe donc pas. En remplaçant ces poids par 1 on obtient une machine déterministe qui utilise le même temps pour calculer la même fonction binaire. L'autre direction est triviale puisqu'une machine dont tous les poids sont 0 ou 1 est une machine de Turing algébrique sur S, peu importe le semi-anneau S. ■

La proposition précédente est également vraie pour un semi-anneau fini.

Le théorème suivant découle immédiatement de la définition de la S-MTAA et des quantificateurs algébriques et de la proposition précédente.

Théorème 73. *Soit S, un semi-anneau sans diviseur de zéro ou un semi-anneau fini, alors une fonction booléenne $f : \{0,1\}^* \to \{0,1\}$ est calculable par une k-S-MTAA dont q_0 est existentiel et qui utilise un temps de calcul dans $O(t(n))$ si et seulement s'il existe une fonction binaire $g \in DTIME(t(n))$ et f peut s'exprimer sous la forme :*

$$f(x) = 1 \Leftrightarrow \exists^S_{w_1} \forall^S_{w_2} \dots Q^S_{w_k} \quad g(x, w_1, w_2, \cdots, w_k) = 1,$$

où le quantificateur Q^S est existentiel si k est impair et universel sinon.

Direction future

Nous croyons que la machine de Turing algébrique alternante est assez intéressante pour mériter une étude méthodique.

Il serait intéressant, par exemple, de démontrer qu'une k-S-MTAA peut calculer plus de fonctions qu'une $(k-1)$-S-MTAA qui utilise la même quantité de ressources. On remarque que pour le cas booléen, cela est équivalent à montrer que $\Sigma^p_i \neq \Sigma^p_{i+1}$.

Dans le but de calquer la hiérarchie polynomiale, il pourrait être intéressant de développer une notion d'oracle pour la S-MTA. La valeur, dans S, retournée par l'oracle pourrait être considérée comme le poids de la transition sortante de l'état question.

Montrer une correspondance entre les machines de Turing algébriques alternantes et les machines de Turing algébriques utilisant un oracle constitue un travail non trivial puisque la notion de complément n'est pas bien définie pour les fonctions non-booléennes

ce qui élimine la possibilité d'adapter directement la preuve classique d'équivalence entre les deux notions.

Une généralisation du théorème 73 aux fonctions à valeur dans S permettrait peut-être de comparer la puissance expressive des S-`MTA` avec oracle à la puissance expressive des S-`MTAA`. Il n'est pas évident que la généralisation du théorème 73 à tous les semi-anneaux S soit possible. En particulier, il semble peu probable que cela soit le cas s'il est possible qu'une machine de Turing algébrique alternante sur S calcule une fonction, dans S, dont les valeurs ne sont pas encodables sur $O(t(n))$ cellules ou si les opérations sur S sont très complexes. On note toutefois que le théorème devient vrai pour tous les semi-anneaux dès qu'on remplace $DTIME$ par $S\text{-}DTIME$ dans l'énoncé.

Chapitre 4

Conclusion

Dans la première partie de ce mémoire, nous avons étudié les programmes de génération sur les semi-anneaux. Pour ce faire, l'analyse a été divisée en deux sections : la puissance expressive et le problème d'évaluation. Nous avons en particulier étudié les programmes de génération sur les treillis et sur les naturels. Dans le premier cas, nous avons présenté une proposition qui une fois restreinte aux chaînes, nous a permis de démontrer que, dans ce cas particulier, les programmes de génération sont équivalents aux formules booléennes en forme normale conjonctive en ce sens qu'ils peuvent calculer les mêmes fonctions. Il serait intéressant de déterminer la puissance expressive des programmes de génération sur les treillis qui ne forment pas une chaîne. Pour ce faire, la proposition 36 pourrait s'avérer très utile. Dans le second cas, nous avons démontré que toutes les fonctions de NL sont calculables par une famille uniforme de programmes de génération sur les naturels. Nous avons également démontré que l'évaluation d'un programme de génération sur les naturels est un problème `NP-complet`. Il serait donc intéressant de découvrir s'il est possible de calculer des fonctions dont l'appartenance à la classe NL est inconnue. En général, il pourrait être intéressant de trouver des semi-anneaux qui capturent une classe de fonctions plus vaste que les semi-anneaux étudiés dans ce mémoire.

Dans la seconde et dernière partie de ce mémoire, nous avons amorcé une étude dédiée à la machine de Turing algébrique. Après avoir adapté quelques résultats classiques traitant des réductions de ressources sans incidences, nous avons démontré l'existence d'une hiérarchie spatiale stricte pour la machine de Turing algébrique sur les semi-anneaux finis. Cette hiérarchie démontre qu'il est possible d'augmenter la puissance

expressive d'une machine de Turing algébrique si on lui permet d'utiliser plus d'espace. Cette hiérarchie est conditionnelle au théorème de Savitch et il serait intéressant de déterminer l'éventail des semi-anneaux pour lesquels la machine de Turing algébrique satisfait ce théorème. Nous avons terminé le chapitre par une discussion sur la machine de Turing algébrique alternante. À défaut de résultats majeurs, nous avons identifié plusieurs avenues intéressantes pour la recherche à venir. En particulier, nous croyons qu'il serait profitable de définir une notion d'oracle pour la machine de Turing algébrique. Il serait alors possible de construire un parallèle entre les machines de Turing algébriques utilisant un oracle et les machines de Turing algébriques alternantes en se basant sur les résultats connus pour la machine de Turing classique.

Liste symboles

Bibliographie

[Ake78] Sheldon B. AKERS : Binary decision diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.

[Bar89] David A. Mix BARRINGTON : Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[Bei96] Amos BEIMEL : *Secure Schemes for Secret Sharing and Key Distribution.* Thèse de doctorat, Technion, 1996.

[BG99] Amos BEIMEL et Anna GAL : On arithmetic branching programs. *Journal of Computer and System Sciences*, 59(2):195–220, 1999.

[BH01] Martin BEAUDRY et Markus HOLZER : The complexity of tensor circuit evaluation. *In MFCS*, pages 173–185, 2001.

[BS81] Stanley BURRIS et Hanamantagouda P. SANKAPPANAVAR : *A Course in Universal Algebra.* Springer Verlag, Berlin, 1981.

[CKS81] Ashok K. CHANDRA, Dexter C. KOZEN et Larry J. STOCKMEYER : Alternation. *J. ACM*, 28(1):114–133, 1981.

[Cob66] Alan COBHAM : The recognition problem for the set of perfect squares. *In Proceedings of the 7th Symposium on Switching and Automata Theory*, pages 78–87, 1966.

[Coo71] Stephen A. COOK : The complexity of theorem-proving procedures. *In STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[DHM02] Carsten DAMM, Markus HOLZER et Pierre MCKENZIE : The complexity of tensor calculus. *Computational Complexity*, 11(1/2):54–89, 2002.

[FFK91] Stephen A. FENNER, Lance FORTNOW et Stuart A. KURTZ : Gap-definable counting classes. *In Structure in Complexity Theory Conference*, pages 30–42, 1991.

[FSS84] Merrick L. FURST, James B. SAXE et Michael SIPSER : Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[Gal95] Anna GAL : *Combinatorial Methods in Boolean Function Complexity.* Thèse de doctorat, University of Chicago, 1995.

[GJ90] Michael R. GAREY et David S. JOHNSON : *Computers and Intractability ; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[HU79] John E. HOPCROFT et Jeffrey D. ULLMAN : *Introduction To Automata Theory, Languages And Computation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979.

[KS86] Werner KUICH et Arto SALOMAA : *Semirings, automata, languages.* Springer-Verlag, London, UK, 1986.

[KW93] M. KARCHMER et A. WIGDERSON : On span programs. *In Proceedings of the 8th Annual Conference on Structure in Complexity Theory*, pages 102–111, San Diego, CA, USA, may 1993. IEEE Computer Society Press.

[Lee59] C. Y. LEE : Representation of switching circuits by binary-decision program. *Bell Systems Technical Journal*, 38:985–999, 1959.

[Ler83] Pierre LEROUX : *Algèbre linéaire, une approche matricielle.* Modulo éditeur, 1983.

[McK04] Pierre MCKENZIE : Notes de cours disponible à l'adresse internet « `http ://www.iro.umontreal.ca/~mckenzie/ift6121/` », Septembre 2004.

[Mei89] Christoph MEINEL : *Modified branching programs and their computational power.* Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[PS98] Pavel PUDLÁK et Jiří SGALL : Algebraic models of computation and interpolation for algebraic proof systems. *Proof Complexity and Feasible Arithmetic*, 39:279–296, 1998.

[Raz91] Alexander A. RAZBOROV : Lower bounds for deterministic and nondeterministic branching programs. *In Fundamentals of Computation Theory*, numéro 529 de Lecture Notes in Computer Science, pages 47–60, New York/Berlin, 1991. Springer Verlag.

[Sch78] Thomas J. SCHAEFER : The complexity of satisfiability problems. *In Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM Press, 1978.

[Sip05] Michael SIPSER : *Introduction to the Theory of Computation.* PWS Publishing Company, 20 Park Plaza, Boston, MA 02116, seconde édition, 2005.

[Tur36] Alan TURING : On computable numbers with applications to the entscheidungsproblem. *In Proceedings of the London Mathematical Society*, volume 42, pages 230–265, 1936.

[Val79] Leslie G. VALIANT : The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Weg00] Ingo WEGENER : *Branching programs and binary decision diagrams : theory and applications.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.